Trusted Platform Module Library
Part 4: Supporting Routines

Family “2.0”
Level 00 Revision 01.50
September 18, 2018

Committee Draft

Contact: admin@trustedcomputinggroup.org

Work in Progress
This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document.

TCG Public Review
Copyright © TCG 2006-2019
Licenses and Notices

Copyright Licenses:

- Trusted Computing Group (TCG) grants to the user of the source code in this specification (the “Source Code”) a worldwide, irrevocable, nonexclusive, royalty free, copyright license to reproduce, create derivative works, distribute, display and perform the Source Code and derivative works thereof, and to grant others the rights granted herein.

- The TCG grants to the user of the other parts of the specification (other than the Source Code) the rights to reproduce, distribute, display, and perform the specification solely for the purpose of developing products based on such documents.

Source Code Distribution Conditions:

- Redistributions of Source Code must retain the above copyright licenses, this list of conditions and the following disclaimers.

- Redistributions in binary form must reproduce the above copyright licenses, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution.

Disclaimers:

- THE COPYRIGHT LICENSES SET FORTH ABOVE DO NOT REPRESENT ANY FORM OF LICENSE OR WAIVER, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, WITH RESPECT TO PATENT RIGHTS HELD BY TCG MEMBERS (OR OTHER THIRD PARTIES) THAT MAY BE NECESSARY TO IMPLEMENT THIS SPECIFICATION OR OTHERWISE. Contact TCG Administration (admin@trustedcomputinggroup.org) for information on specification licensing rights available through TCG membership agreements.

- THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO EXPRESS OR IMPLIED WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE, ACCURACY, COMPLETENESS, OR NONINFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

- Without limitation, TCG and its members and licensors disclaim all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

Any marks and brands contained herein are the property of their respective owners.
## CONTENTS

1 Scope ........................................................................................................................................... 1
2 Terms and definitions .................................................................................................................... 1
3 Symbols and abbreviated terms ..................................................................................................... 1
4 Automation ..................................................................................................................................... 1
   4.1 Configuration Parser ................................................................................................................ 1
   4.2 Structure Parser ....................................................................................................................... 2
       4.2.1 Introduction .................................................................................................................... 2
       4.2.2 Unmarshaling Code Prototype ....................................................................................... 2
       4.2.2.1 Simple Types and Structures .................................................................................... 2
       4.2.2.2 Union Types .............................................................................................................. 3
       4.2.2.3 Null Types .............................................................................................................. 3
       4.2.2.4 Arrays ....................................................................................................................... 3
       4.2.3 Marshaling Code Function Prototypes .............................................................................. 4
           4.2.3.1 Simple Types and Structures .................................................................................... 4
           4.2.3.2 Union Types ............................................................................................................ 4
           4.2.3.3 Arrays .................................................................................................................... 4
   4.3 Part 3 Parsing .......................................................................................................................... 5
   4.4 Function Prototypes ............................................................................................................... 5
   4.5 Portability .............................................................................................................................. 6
5 Header Files .................................................................................................................................. 7
   5.1 Introduction ................................................................................................................................ 7
   5.2 BaseTypes.h ............................................................................................................................. 7
   5.3 Capabilities.h .......................................................................................................................... 8
   5.4 CommandAttributeData.h ......................................................................................................... 9
   5.5 CommandAttributes.h ............................................................................................................ 23
   5.6 CommandDispatchData.h .......................................................................................................... 24
   5.7 Commands.h .......................................................................................................................... 91
   5.8 CompilerDependencies.h ......................................................................................................... 98
   5.9 Global.h ................................................................................................................................... 100
      5.9.1 Description ....................................................................................................................... 100
      5.9.2 Includes ............................................................................................................................ 100
      5.9.3 Loaded Object Structures ............................................................................................... 101
          5.9.3.1 Description .............................................................................................................. 101
          5.9.3.2 OBJECT_ATTRIBUTES ......................................................................................... 101
          5.9.3.3 OBJECT Structure .............................................................................................. 102
          5.9.3.4 HASH_OBJECT Structure ................................................................................... 102
          5.9.3.5 ANY_OBJECT ....................................................................................................... 103
      5.9.4 AUTH_DUP Types ............................................................................................................. 103
      5.9.5 Active Session Context .................................................................................................. 103
          5.9.5.1 Description ............................................................................................................. 103
          5.9.5.2 SESSION_ATTRIBUTES ......................................................................................... 103
          5.9.5.3 SESSION Structure .............................................................................................. 104
      5.9.6 PCR ................................................................................................................................. 105
          5.9.6.1 PCR_SAVE Structure ............................................................................................. 105
          5.9.6.2 PCR_POLICY ......................................................................................................... 106
          5.9.6.3 PCR_AUTHVALUE ............................................................................................... 106
      5.9.7 STARTUP_TYPE .................................................................................................................. 106
      5.9.8 NV ..................................................................................................................................... 106
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.9.8.1</td>
<td>NV_INDEX</td>
<td>106</td>
</tr>
<tr>
<td>5.9.8.2</td>
<td>NV_REF</td>
<td>107</td>
</tr>
<tr>
<td>5.9.8.3</td>
<td>NV_PIN</td>
<td>107</td>
</tr>
<tr>
<td>5.9.9</td>
<td>COMMIT_INDEX_MASK</td>
<td>107</td>
</tr>
<tr>
<td>5.9.10</td>
<td>RAM Global Values</td>
<td>107</td>
</tr>
<tr>
<td>5.9.10.1</td>
<td>Description</td>
<td>107</td>
</tr>
<tr>
<td>5.9.10.2</td>
<td>g_rclIndex</td>
<td>108</td>
</tr>
<tr>
<td>5.9.10.3</td>
<td>g_exclusiveAuditSession</td>
<td>108</td>
</tr>
<tr>
<td>5.9.10.4</td>
<td>g_time</td>
<td>108</td>
</tr>
<tr>
<td>5.9.10.5</td>
<td>g_timeEpoch</td>
<td>108</td>
</tr>
<tr>
<td>5.9.10.6</td>
<td>g_phEnable</td>
<td>108</td>
</tr>
<tr>
<td>5.9.10.7</td>
<td>g_pcrReConfig</td>
<td>108</td>
</tr>
<tr>
<td>5.9.10.8</td>
<td>g_DRTMHandle</td>
<td>108</td>
</tr>
<tr>
<td>5.9.10.9</td>
<td>g_DrtmPreStartup</td>
<td>108</td>
</tr>
<tr>
<td>5.9.10.10</td>
<td>g_StartupLocality3</td>
<td>109</td>
</tr>
<tr>
<td>5.9.10.11</td>
<td>TPM_SU_NONE</td>
<td>109</td>
</tr>
<tr>
<td>5.9.10.12</td>
<td>TPM_SU_DA_USED</td>
<td>109</td>
</tr>
<tr>
<td>5.9.10.13</td>
<td>Startup Flags</td>
<td>109</td>
</tr>
<tr>
<td>5.9.10.14</td>
<td>g_daUsed</td>
<td>109</td>
</tr>
<tr>
<td>5.9.10.15</td>
<td>g_updateNV</td>
<td>110</td>
</tr>
<tr>
<td>5.9.10.16</td>
<td>g_powerWasLost</td>
<td>110</td>
</tr>
<tr>
<td>5.9.10.17</td>
<td>g_clearOrderly</td>
<td>110</td>
</tr>
<tr>
<td>5.9.10.18</td>
<td>g_prevOrderlyState</td>
<td>110</td>
</tr>
<tr>
<td>5.9.10.19</td>
<td>g_nvOk</td>
<td>110</td>
</tr>
<tr>
<td>5.9.10.20</td>
<td>g_platformUnique</td>
<td>111</td>
</tr>
<tr>
<td>5.9.11</td>
<td>Persistent Global Values</td>
<td>111</td>
</tr>
<tr>
<td>5.9.11.1</td>
<td>Description</td>
<td>111</td>
</tr>
<tr>
<td>5.9.11.2</td>
<td>PERSISTENT_DATA</td>
<td>111</td>
</tr>
<tr>
<td>5.9.11.3</td>
<td>ORDERLY_DATA</td>
<td>113</td>
</tr>
<tr>
<td>5.9.11.4</td>
<td>STATE_CLEAR_DATA</td>
<td>114</td>
</tr>
<tr>
<td>5.9.11.5</td>
<td>State Reset Data</td>
<td>115</td>
</tr>
<tr>
<td>5.9.12</td>
<td>NV Layout</td>
<td>116</td>
</tr>
<tr>
<td>5.9.13</td>
<td>Global Macro Definitions</td>
<td>117</td>
</tr>
<tr>
<td>5.9.14</td>
<td>Private data</td>
<td>118</td>
</tr>
<tr>
<td>5.10</td>
<td>GpMacros.h</td>
<td>123</td>
</tr>
<tr>
<td>5.10.1</td>
<td>Introduction</td>
<td>123</td>
</tr>
<tr>
<td>5.10.2</td>
<td>For Self-test</td>
<td>123</td>
</tr>
<tr>
<td>5.10.3</td>
<td>For Failures</td>
<td>123</td>
</tr>
<tr>
<td>5.10.4</td>
<td>Derived from Vendor-specific values</td>
<td>124</td>
</tr>
<tr>
<td>5.10.5</td>
<td>Compile-time Checks</td>
<td>124</td>
</tr>
<tr>
<td>5.11</td>
<td>InternalRoutines.h</td>
<td>128</td>
</tr>
<tr>
<td>5.12</td>
<td>LibSupport.h</td>
<td>130</td>
</tr>
<tr>
<td>5.13</td>
<td>NV.h</td>
<td>132</td>
</tr>
<tr>
<td>5.13.1</td>
<td>Index Type Definitions</td>
<td>132</td>
</tr>
<tr>
<td>5.13.2</td>
<td>Attribute Macros</td>
<td>132</td>
</tr>
<tr>
<td>5.13.3</td>
<td>Orderly RAM Values</td>
<td>133</td>
</tr>
<tr>
<td>5.14</td>
<td>PRNG_TestVectors.h</td>
<td>135</td>
</tr>
<tr>
<td>5.15</td>
<td>SelfTest.h</td>
<td>136</td>
</tr>
<tr>
<td>5.15.1</td>
<td>Introduction</td>
<td>136</td>
</tr>
<tr>
<td>5.15.2</td>
<td>Defines</td>
<td>136</td>
</tr>
<tr>
<td>5.16</td>
<td>SupportLibraryFunctionPrototypes_fp.h</td>
<td>138</td>
</tr>
<tr>
<td>5.16.1</td>
<td>Introduction</td>
<td>138</td>
</tr>
</tbody>
</table>
Part 4: Supporting Routines

6 Main

6.1 Introduction
6.2 ExecCommand.c
6.3 CommandDispatcher.c

6.4 SessionProcess.c

6.4.1 Introduction
6.4.2 Includes and Data Definitions
6.4.3 Authorization Support Functions

6.4.4 Session Parsing Functions

5.16.2 SupportLibInit() .................................. 138
5.16.3 MathLibraryCompatibilityCheck() ............. 138
5.16.4 BnModMult() ..................................... 138
5.16.5 BnMult() ........................................ 138
5.16.6 BnDiv() .......................................... 138
5.16.7 BnMod() .......................................... 139
5.16.8 BnGcd() ......................................... 139
5.16.9 BnModExp() ...................................... 139
5.16.10 BnModInverse() ................................ 139
5.16.11 BnEccModMult() ................................ 139
5.16.12 BnEccModMult2() ................................ 139
5.16.13 BnEccAdd() .................................... 139
5.16.14 BnCurveInitialize() ............................ 140

5.17 TPMB.h ............................................... 141
5.18 Tpm.h ................................................. 142
5.19 TpmBuildSwitches.h .................................. 143
5.20 TpmError.h .......................................... 148
5.21 TpmTypes.h ......................................... 149
5.22 VendorString.h ...................................... 178
5.23 swap.h ............................................... 179
6.4.5.10 CheckSessionHMAC() ........................................... 207
6.4.5.11 CheckPolicyAuthSession() .................................... 208
6.4.5.12 RetrieveSessionData() ......................................... 211
6.4.5.13 CheckLockedOut() ............................................ 213
6.4.5.14 CheckAuthSession() ........................................... 214
6.4.5.15 CheckCommandAudit() ......................................... 217
6.4.5.16 ParseSessionBuffer() .......................................... 217
6.4.5.17 CheckAuthNoSession() ......................................... 220

6.4.5 Response Session Processing ........................................ 220

6.4.5.1 Introduction ................................................................ 220
6.4.5.2 ComputeRpHash() ................................................. 220
6.4.5.3 InitAuditSession() ............................................... 221
6.4.5.4 UpdateAuditDigest ............................................. 221
6.4.5.5 Audit() ................................................................... 222
6.4.5.6 CommandAudit() .................................................. 222
6.4.5.7 UpdateAuditSessionStatus() .................................. 223
6.4.5.8 ComputeResponseHMAC() ..................................... 224
6.4.5.9 UpdateInternalSession() ..................................... 225
6.4.5.10 BuildSingleResponseAuth() ................................. 225
6.4.5.11 UpdateAllNonceTPM() ....................................... 226
6.4.5.12 BuildResponseSession() ....................................... 226
6.4.5.13 SessionRemoveAssociationToHandle() ..................... 228

7 Command Support Functions ............................................... 229

7.1 Introduction ..................................................................... 229
7.2 Attestation Command Support (Attest_spt.c) ..................... 229

7.2.1 Includes ................................................................... 229
7.2.2 Functions .................................................................. 229

7.2.2.1 FillInAttestInfo() ............................................... 229
7.2.2.2 SignAttestInfo() .................................................. 230
7.2.2.3 IsSigningObject() .............................................. 231

7.3 Context Management Command Support (Context_spt.c) .... 232

7.3.1 Includes ................................................................... 232
7.3.2 Functions .................................................................. 232

7.3.2.1 ComputeContextProtectionKey() ........................... 232
7.3.2.2 ComputeContextIntegrity() .................................. 233
7.3.2.3 SequenceDataExport() ....................................... 234
7.3.2.4 SequenceDataImport() ....................................... 234

7.4 Policy Command Support (Policy_spt.c) ............................ 235

7.4.1 Includes ................................................................... 235
7.4.2 Functions .................................................................. 235

7.4.2.1 PolicyParameterChecks() .................................... 235
7.4.2.2 PolicyContextUpdate() ....................................... 236
7.4.2.3 ComputeAuthTimeout() ....................................... 237
7.4.2.4 PolicyDigestClear() ........................................... 237

7.5 NV Command Support (NV_spt.c) .................................... 240

7.5.1 Includes ................................................................... 240
7.5.2 Functions .................................................................. 240

7.5.2.1 NvReadAccessChecks() ....................................... 240
7.5.2.2 NvWriteAccessChecks() ....................................... 241
7.5.2.3 NvClearOrderly() .............................................. 241
7.5.2.4 NvIsPinPassIndex() ........................................... 242
7.6 Object Command Support (Object_spt.c) ................................................................. 243
  7.6.1 Includes .................................................................................................................. 243
  7.6.2 Local Functions ..................................................................................................... 243
    7.6.2.1 GetIv2BSize() ............................................................................................... 243
    7.6.2.2 ComputeProtectionKeyParms() ..................................................................... 243
    7.6.2.3 ComputeOuterIntegrity() ............................................................................... 244
    7.6.2.4 ComputeInnerIntegrity() ............................................................................... 245
    7.6.2.5 ProduceInnerIntegrity() ............................................................................... 245
    7.6.2.6 CheckInnerIntegrity() .................................................................................. 246
  7.6.3 Public Functions .................................................................................................... 247
    7.6.3.1 AdjustAuthSize() ......................................................................................... 247
    7.6.3.2 AreAttributesForParent() ............................................................................ 247
    7.6.3.3 CreateChecks() ............................................................................................. 247
    7.6.3.4 SchemeChecks .............................................................................................. 248
    7.6.3.5 PublicAttributesValidation() ....................................................................... 252
    7.6.3.6 FillInCreationData() ..................................................................................... 253
    7.6.3.7 GetSeedForKDF() ......................................................................................... 254
    7.6.3.8 ProduceOuterWrap() .................................................................................... 255
    7.6.3.9 UnwrapOuter() .............................................................................................. 256
    7.6.3.10 MarshalSensitive() ..................................................................................... 257
    7.6.3.11 SensitiveToPrivate() ................................................................................... 258
    7.6.3.12 PrivateToSensitive() ................................................................................... 259
    7.6.3.13 SensitiveToDuplicate() .............................................................................. 260
    7.6.3.14 DuplicateToSensitive() .............................................................................. 262
    7.6.3.15 SecretToCredential() .................................................................................. 264
    7.6.3.16 CredentialToSecret() .................................................................................. 264
    7.6.3.17 MemoryRemoveTrailingZeros() .................................................................. 265
    7.6.3.18 SetLabelAndContext() ............................................................................... 266
    7.6.3.19 UnmarshalToPublic() .................................................................................. 266
    7.6.3.20 ObjectSetExternal() .................................................................................... 267
  7.7 Encrypt Decrypt Support (EncryptDecrypt_spt.c) ......................................................... 268
8 Subsystem ...................................................................................................................... 270
  8.1 CommandAudit.c ....................................................................................................... 270
    8.1.1 Introduction ........................................................................................................ 270
    8.1.2 Includes .............................................................................................................. 270
    8.1.3 Functions ........................................................................................................... 270
      8.1.3.1 CommandAuditPreInstall_Init() ................................................................. 270
      8.1.3.2 CommandAuditStartup() ............................................................................. 270
      8.1.3.3 CommandAuditSet() ................................................................................... 271
      8.1.3.4 CommandAuditClear() ............................................................................... 271
      8.1.3.5 CommandAuditIsRequired() ....................................................................... 272
      8.1.3.6 CommandAuditCapGetCList() ................................................................... 272
      8.1.3.7 CommandAuditGetDigest ............................................................................ 273
  8.2 DA.c ............................................................................................................................ 275
    8.2.1 Introduction ........................................................................................................ 275
    8.2.2 Includes and Data Definitions ............................................................................ 275
    8.2.3 Functions ........................................................................................................... 275
      8.2.3.1 DAPreInstall_Init() ..................................................................................... 275
      8.2.3.2 DASTartup() ............................................................................................... 275
      8.2.3.3 DAREgisterFailure() .................................................................................. 276
      8.2.3.4 DASelfHeal() ............................................................................................ 277
  8.3 Hierarchy.c .................................................................................................................. 279
8.3.1 Introduction ................................................................. 279
8.3.2 Includes ........................................................................ 279
8.3.3 Functions ....................................................................... 279
  8.3.3.1 HierarchyPreInstall() .................................................. 279
  8.3.3.2 HierarchyStartup() ....................................................... 280
  8.3.3.3 HierarchyGetProof() ..................................................... 280
  8.3.3.4 HierarchyGetPrimarySeed() ........................................ 281
  8.3.3.5 HierarchyIsEnabled() ................................................... 281
8.4 NvDynamic.c ...................................................................... 283
  8.4.1 Introduction ................................................................... 283
  8.4.2 Includes, Defines and Data Definitions ....................... 283
  8.4.3 Local Functions .............................................................. 283
    8.4.3.1 NvNext() .................................................................. 283
    8.4.3.2 NvNextByType() ......................................................... 284
    8.4.3.3 NvNextIndex() ............................................................. 284
    8.4.3.4 NvNextEvict() ............................................................. 285
    8.4.3.5 NvGetEnd() ................................................................ 285
    8.4.3.6 NvGetFreeBytes .............................................................. 285
    8.4.3.7 NvTestSpace() ............................................................... 285
    8.4.3.8 NvWriteNvListEnd() .................................................... 286
    8.4.3.9 NvAdd() ..................................................................... 287
    8.4.3.10 NvDelete() ................................................................. 288
  8.4.4 RAM-based NV Index Data Access Functions .................. 289
    8.4.4.1 Introduction ................................................................. 289
    8.4.4.2 NvRamNext() ............................................................... 289
    8.4.4.3 NvRamGetEnd() ............................................................. 289
    8.4.4.4 NvRamTestSpaceIndex() ............................................. 290
    8.4.4.5 NvRamGetIndex() .......................................................... 290
    8.4.4.6 NvUpdateIndexOrderlyData() ...................................... 290
    8.4.4.7 NvAddRAM() ............................................................... 291
    8.4.4.8 NvDeleteRAM() ............................................................ 291
    8.4.4.9 NvReadIndex() ............................................................. 292
    8.4.4.10 NvReadObject() .......................................................... 292
    8.4.4.11 NvFindEvict() ............................................................. 292
    8.4.4.12 NvIndexIsDefined() .................................................... 293
    8.4.4.13 NvConditionallyWrite() ............................................. 293
    8.4.4.14 NvReadNvIndexAttributes() ...................................... 294
    8.4.4.15 NvReadRamIndexAttributes() ..................................... 294
    8.4.4.16 NvWriteNvIndexAttributes() ...................................... 294
    8.4.4.17 NvWriteRamIndexAttributes() ..................................... 294
  8.4.5 Externally Accessible Functions .................................... 295
    8.4.5.1 NvIsPlatformPersistentHandle() .................................. 295
    8.4.5.2 NvIsOwnerPersistentHandle() ...................................... 295
    8.4.5.3 NvIndexIsAccessible() .................................................. 295
    8.4.5.4 NvGetEvictObject() ....................................................... 296
    8.4.5.5 NvIndexCacheInit() ....................................................... 297
    8.4.5.6 NvGetIndexData() ........................................................ 297
    8.4.5.7 NvGetUINT64Data() ...................................................... 298
    8.4.5.8 NvWriteIndexAttributes() ............................................ 298
    8.4.5.9 NvWriteIndexAuth() ..................................................... 299
    8.4.5.10 NvGetIndexInfo() ........................................................ 299
    8.4.5.11 NvWriteIndexData() ..................................................... 300
    8.4.5.12 NvWriteUINT64Data() ................................................. 301
    8.4.5.13 NvGetIndexName() ...................................................... 302
8.4.6.1 Introduction ........................................ 313
8.4.6.2 NvReadMaxCount() .................................. 314
8.4.6.3 NvUpdateMaxCount() ................................ 314
8.4.6.4 NvSetMaxCount() ................................... 314
8.4.6.5 NvGetMaxCount() ................................... 314

8.5 NvReserved.c ........................................... 316
8.5.1 Introduction ........................................... 316
8.5.2 Includes, Defines ..................................... 316
8.5.3 Functions ............................................. 316
  8.5.3.1 NvInitStatic() ...................................... 316
  8.5.3.2 NvCheckState() ..................................... 317
  8.5.3.3 NvCommit ........................................... 317
  8.5.3.4 NvPowerOn() ....................................... 317
  8.5.3.5 NvManufacture() ..................................... 318
  8.5.3.6 NvRead() .......................................... 318
  8.5.3.7 NvWrite() ......................................... 318
  8.5.3.8 NvUpdatePersistent() ............................... 319
  8.5.3.9 NvClearPersistent() ................................. 319
  8.5.3.10 NvReadPersistent() ................................. 319

8.6 Object.c ................................................. 320
8.6.1 Introduction .......................................... 320
8.6.2 Includes and Data Definitions ........................ 320
8.6.3 Functions .............................................. 320
  8.6.3.1 ObjectFlush() ....................................... 320
  8.6.3.2 ObjectSetInUse() .................................... 320
  8.6.3.3 ObjectStartup() ...................................... 320
  8.6.3.4 ObjectCleanupEvict() ............................... 321
  8.6.3.5 IsObjectPresent() .................................... 321
  8.6.3.6 ObjectIsSequence() .................................. 321
  8.6.3.7 HandleToObject() ..................................... 322
  8.6.3.8 ObjectGetNameAlg() .................................. 322
  8.6.3.9 GetQualifiedName() .................................. 322
  8.6.3.10 ObjectGetHierarchy() ............................... 323
  8.6.3.11 GetHierarchy() ..................................... 324
  8.6.3.12 FindEmptyObjectSlot() ............................ 324
8.7 PCR.c.................................................................................337
8.7.1 Introduction ........................................................................337
8.7.2 Includes, Defines, and Data Definitions .........................337
8.7.3 Functions ...........................................................................337
  8.7.3.1 PCRBelongsToAuthGroup()...........................................337
  8.7.3.2 PCRBelongsToPolicyGroup()...........................................338
  8.7.3.3 PCRBelongsToTCGroup()...................................................338
  8.7.3.4 PCRPolicyIsAvailable()..................................................339
  8.7.3.5 PCRGetAuthValue().........................................................339
  8.7.3.6 PCRGetAuthPolicy().......................................................340
  8.7.3.7 PCRSetStart().................................................................340
  8.7.3.8 GetSavedPcrPointer().....................................................341
  8.7.3.9 PcrIsAllocated()..............................................................342
  8.7.3.10 GetPcrPointer()..............................................................342
  8.7.3.11 IsPcrSelected()..............................................................343
  8.7.3.12 FilterPcr()..................................................................343
  8.7.3.13 PcrDrtm()..................................................................344
  8.7.3.14 PCR_ClearAuth()..........................................................344
  8.7.3.15 PCRStartup()..............................................................345
  8.7.3.16 PCRStateSave()............................................................346
  8.7.3.17 PCRIsStateSaved().......................................................347
  8.7.3.18 PCRIsResetAllowed()..................................................347
  8.7.3.19 PCRChanged()..............................................................348
  8.7.3.20 PCRIsExtendAllowed()................................................348
  8.7.3.21 PCRExtend()...............................................................349
  8.7.3.22 PCRComputeCurrentDigest().......................................349
  8.7.3.23 PCRRead()..................................................................350
  8.7.3.24 PcrWrite()..................................................................351
  8.7.3.25 PCRAllocate()..............................................................352
  8.7.3.26 PCRSetValue().............................................................353
  8.7.3.27 PCRResetDynamics......................................................354
  8.7.3.28 PCRGetAllocation().....................................................355
  8.7.3.29 PCRSetSelectBit().........................................................355
  8.7.3.30 PCRGetProperty().........................................................355
  8.7.3.31 PCRCapGetProperties().................................................357
  8.7.3.32 PCRCapGetHandles()....................................................358

8.8 PP.c .......................................................................................360
8.8.1 Introduction ....................................................................................360
8.8.2 Includes..........................................................................................360
8.8.3 Functions ......................................................................................360
  8.8.3.1 PhysicalPresencePreInstall_Init() ...........................................360
  8.8.3.2 PhysicalPresenceCommandSet() ............................................360
  8.8.3.3 PhysicalPresenceCommandClear() ..........................................361
  8.8.3.4 PhysicalPresenceIsRequired() ................................................361
  8.8.3.5 PhysicalPresenceCapGetCCList() ..........................................361

8.9 Session.c ..........................................................................................363
  8.9.1 Introduction ..................................................................................363
  8.9.2 Includes, Defines, and Local Variables ........................................364
  8.9.3 File Scope Function -- ContextIdSetOldest() ..............................364
  8.9.4 Startup Function -- SessionStartup() .........................................365
  8.9.5 Access Functions ........................................................................365
    8.9.5.1 SessionIsLoaded() .................................................................365
    8.9.5.2 SessionIsSaved() ................................................................366
    8.9.5.3 SequenceNumberForSavedContextIsValid() .....................367
    8.9.5.4 SessionPCRValuesCurrent() ..............................................367
    8.9.5.5 SessionGet() ....................................................................367
  8.9.6 Utility Functions ..........................................................................368
    8.9.6.1 ContextIdSessionCreate() .................................................368
    8.9.6.2 SessionCreate() ..................................................................369
    8.9.6.3 SessionContextSave() ..........................................................371
    8.9.6.4 SessionContextLoad() ..........................................................372
    8.9.6.5 SessionFlush() ..................................................................374
    8.9.6.6 SessionComputeBoundEntity() ..........................................374
    8.9.6.7 SessionSetStartTime() .........................................................375
    8.9.6.8 SessionResetPolicyData() ..................................................375
    8.9.6.9 SessionCapGetLoaded() ......................................................376
    8.9.6.10 SessionCapGetSaved() ......................................................377
    8.9.6.11 SessionCapGetLoadedNumber() ......................................378
    8.9.6.12 SessionCapGetLoadedAvail() ..........................................378
    8.9.6.13 SessionCapGetActiveNumber() ......................................378
    8.9.6.14 SessionCapGetActiveAvail() .........................................379

8.10 Time.c .............................................................................................380
  8.10.1 Introduction .................................................................................380
  8.10.2 Includes.......................................................................................380
  8.10.3 Functions ....................................................................................380
    8.10.3.1 TimePowerOn() ................................................................380
    8.10.3.2 TimeNewEpoch() ...............................................................380
    8.10.3.3 TimeStartup() ....................................................................380
    8.10.3.4 TimeClockUpdate() ............................................................381
    8.10.3.5 TimeUpdate() .....................................................................381
    8.10.3.6 TimeUpdateToCurrent() ......................................................382
    8.10.3.7 TimeSetAdjustRate() ...........................................................382
    8.10.3.8 TimeGetMarshaled() ..........................................................383
    8.10.3.9 TimeFillInfo .......................................................................383

9 Support ...............................................................................................385
  9.1 AlgorithmCap.c .................................................................................385
    9.1.1 Description ...............................................................................385
    9.1.2 Includes and Defines .................................................................385
    9.1.3 AlgorithmCapGetImplemented() .............................................387
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.4</td>
<td>AlgorithmGetImplementedVector() ................................................................. 387</td>
</tr>
<tr>
<td>9.2</td>
<td>Bits.c .............................................................................................................. 389</td>
</tr>
<tr>
<td>9.2.1</td>
<td>Introduction .................................................................................................... 389</td>
</tr>
<tr>
<td>9.2.2</td>
<td>Includes .......................................................................................................... 389</td>
</tr>
<tr>
<td>9.2.3</td>
<td>Functions ........................................................................................................ 389</td>
</tr>
<tr>
<td>9.2.3.1</td>
<td>TestBit() ......................................................................................................... 389</td>
</tr>
<tr>
<td>9.2.3.2</td>
<td>SetBit() .......................................................................................................... 389</td>
</tr>
<tr>
<td>9.2.3.3</td>
<td>ClearBit() ....................................................................................................... 389</td>
</tr>
<tr>
<td>9.3</td>
<td>CommandCodeAttributes.c ................................................................................. 391</td>
</tr>
<tr>
<td>9.3.1</td>
<td>Introduction .................................................................................................... 391</td>
</tr>
<tr>
<td>9.3.2</td>
<td>Includes and Defines ....................................................................................... 391</td>
</tr>
<tr>
<td>9.3.3</td>
<td>Command Attribute Functions ........................................................................... 391</td>
</tr>
<tr>
<td>9.3.3.1</td>
<td>NextImplementedIndex() .................................................................................. 391</td>
</tr>
<tr>
<td>9.3.3.2</td>
<td>GetClosestCommandIndex() .............................................................................. 392</td>
</tr>
<tr>
<td>9.3.3.3</td>
<td>CommandCodeToCommandIndex() ...................................................................... 394</td>
</tr>
<tr>
<td>9.3.3.4</td>
<td>GetNextCommandIndex() .................................................................................. 395</td>
</tr>
<tr>
<td>9.3.3.5</td>
<td>GetCommandCode() .......................................................................................... 395</td>
</tr>
<tr>
<td>9.3.3.6</td>
<td>CommandAuthRole() ........................................................................................ 396</td>
</tr>
<tr>
<td>9.3.3.7</td>
<td>EncryptSize() .................................................................................................. 396</td>
</tr>
<tr>
<td>9.3.3.8</td>
<td>DecryptSize() ................................................................................................ 397</td>
</tr>
<tr>
<td>9.3.3.9</td>
<td>IsSessionAllowed() .......................................................................................... 397</td>
</tr>
<tr>
<td>9.3.3.10</td>
<td>IsHandleInResponse() ...................................................................................... 397</td>
</tr>
<tr>
<td>9.3.3.11</td>
<td>IsWriteOperation() .......................................................................................... 397</td>
</tr>
<tr>
<td>9.3.3.12</td>
<td>IsReadOperation() ............................................................................................ 398</td>
</tr>
<tr>
<td>9.3.3.13</td>
<td>CommandCapGetCCList() .................................................................................. 399</td>
</tr>
<tr>
<td>9.3.3.14</td>
<td>IsVendorCommand() .......................................................................................... 399</td>
</tr>
<tr>
<td>9.4</td>
<td>Entity.c ............................................................................................................. 401</td>
</tr>
<tr>
<td>9.4.1</td>
<td>Description ...................................................................................................... 401</td>
</tr>
<tr>
<td>9.4.2</td>
<td>Includes .......................................................................................................... 401</td>
</tr>
<tr>
<td>9.4.3</td>
<td>Functions ........................................................................................................ 401</td>
</tr>
<tr>
<td>9.4.3.1</td>
<td>EntityGetLoadStatus() .................................................................................... 401</td>
</tr>
<tr>
<td>9.4.3.2</td>
<td>EntityGetAuthValue() ...................................................................................... 402</td>
</tr>
<tr>
<td>9.4.3.3</td>
<td>EntityGetAuthPolicy() ..................................................................................... 405</td>
</tr>
<tr>
<td>9.4.3.4</td>
<td>EntityGetName() ............................................................................................... 406</td>
</tr>
<tr>
<td>9.4.3.5</td>
<td>EntityGetHierarchy() ...................................................................................... 407</td>
</tr>
<tr>
<td>9.5</td>
<td>Global.c ........................................................................................................... 409</td>
</tr>
<tr>
<td>9.5.1</td>
<td>Description ...................................................................................................... 409</td>
</tr>
<tr>
<td>9.5.2</td>
<td>Includes and Defines ....................................................................................... 409</td>
</tr>
<tr>
<td>9.5.3</td>
<td><strong>IGNORE_STATE</strong> ............................................................................................. 409</td>
</tr>
<tr>
<td>9.5.4</td>
<td>Global Data Values .......................................................................................... 409</td>
</tr>
<tr>
<td>9.5.5</td>
<td>Private Values ................................................................................................ 410</td>
</tr>
<tr>
<td>9.5.5.1</td>
<td>Used in SessionProcess.c ................................................................................ 410</td>
</tr>
<tr>
<td>9.5.5.2</td>
<td>Used in DA.c .................................................................................................... 410</td>
</tr>
<tr>
<td>9.5.5.3</td>
<td>Used in NV.c .................................................................................................... 410</td>
</tr>
<tr>
<td>9.5.5.4</td>
<td>Used in Object.c .............................................................................................. 410</td>
</tr>
<tr>
<td>9.5.5.5</td>
<td>Used in PCR.c .................................................................................................. 410</td>
</tr>
<tr>
<td>9.5.5.6</td>
<td>Used in Session.c ............................................................................................ 410</td>
</tr>
<tr>
<td>9.5.5.7</td>
<td>Used in MemoryLib.c ........................................................................................ 410</td>
</tr>
<tr>
<td>9.5.5.8</td>
<td>used in TmpFail.c ............................................................................................ 411</td>
</tr>
<tr>
<td>9.5.5.9</td>
<td>Used in CryptRand.c ......................................................................................... 411</td>
</tr>
<tr>
<td>9.5.5.10</td>
<td>Used in Manufacture.c ..................................................................................... 411</td>
</tr>
<tr>
<td>9.5.5.11</td>
<td>Used in Power.c ............................................................................................... 411</td>
</tr>
<tr>
<td>9.5.5.12 Purpose-specific String Constants</td>
<td>411</td>
</tr>
<tr>
<td>-----------------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>9.6 Handle.c</td>
<td>412</td>
</tr>
<tr>
<td>9.6.1 Description</td>
<td>412</td>
</tr>
<tr>
<td>9.6.2 Includes</td>
<td>412</td>
</tr>
<tr>
<td>9.6.3 Functions</td>
<td>412</td>
</tr>
<tr>
<td>9.6.3.1 HandleGetType()</td>
<td>412</td>
</tr>
<tr>
<td>9.6.3.2 NextPermanentHandle()</td>
<td>412</td>
</tr>
<tr>
<td>9.6.3.3 PermanentCapGetHandles()</td>
<td>413</td>
</tr>
<tr>
<td>9.6.3.4 PermanentHandleGetPolicy()</td>
<td>413</td>
</tr>
<tr>
<td>9.7 IoBuffers.c</td>
<td>415</td>
</tr>
<tr>
<td>9.7.1 Includes and Data Definitions</td>
<td>415</td>
</tr>
<tr>
<td>9.7.2 Buffers and Functions</td>
<td>415</td>
</tr>
<tr>
<td>9.7.2.1 MemoryIoBufferAllocationReset()</td>
<td>415</td>
</tr>
<tr>
<td>9.7.2.2 MemoryIoBufferZero()</td>
<td>415</td>
</tr>
<tr>
<td>9.7.2.3 MemoryGetInBuffer()</td>
<td>415</td>
</tr>
<tr>
<td>9.7.2.4 MemoryGetOutBuffer()</td>
<td>416</td>
</tr>
<tr>
<td>9.7.2.5 IsLabelProperlyFormatted()</td>
<td>416</td>
</tr>
<tr>
<td>9.8 Locality.c</td>
<td>417</td>
</tr>
<tr>
<td>9.8.1 Includes</td>
<td>417</td>
</tr>
<tr>
<td>9.8.2 LocalityGetAttributes()</td>
<td>417</td>
</tr>
<tr>
<td>9.9 Manufacture.c</td>
<td>418</td>
</tr>
<tr>
<td>9.9.1 Description</td>
<td>418</td>
</tr>
<tr>
<td>9.9.2 Includes and Data Definitions</td>
<td>418</td>
</tr>
<tr>
<td>9.9.3 Functions</td>
<td>418</td>
</tr>
<tr>
<td>9.9.3.1 TPM_Manufacture()</td>
<td>418</td>
</tr>
<tr>
<td>9.9.3.2 TPM_TearDown()</td>
<td>419</td>
</tr>
<tr>
<td>9.9.3.3 TpmEndSimulation()</td>
<td>420</td>
</tr>
<tr>
<td>9.10 Marshal.c</td>
<td>421</td>
</tr>
<tr>
<td>9.10.1 Introduction</td>
<td>421</td>
</tr>
<tr>
<td>9.10.2 Unmarshal and Marshal a Value</td>
<td>421</td>
</tr>
<tr>
<td>9.10.3 Unmarshal and Marshal a Union</td>
<td>422</td>
</tr>
<tr>
<td>9.10.4 Unmarshal and Marshal a Structure</td>
<td>424</td>
</tr>
<tr>
<td>9.10.5 Unmarshal and Marshal an Array</td>
<td>425</td>
</tr>
<tr>
<td>9.10.6 TPM2B Handling</td>
<td>427</td>
</tr>
<tr>
<td>9.11 MathOnByteBuffers.c</td>
<td>428</td>
</tr>
<tr>
<td>9.11.1 Introduction</td>
<td>428</td>
</tr>
<tr>
<td>9.11.2 Functions</td>
<td>428</td>
</tr>
<tr>
<td>9.11.2.1 UnsignedCmpB</td>
<td>428</td>
</tr>
<tr>
<td>9.11.2.2 SignedCompareB()</td>
<td>428</td>
</tr>
<tr>
<td>9.11.2.3 ModExpB</td>
<td>429</td>
</tr>
<tr>
<td>9.11.2.4 DivideB()</td>
<td>430</td>
</tr>
<tr>
<td>9.11.2.5 AdjustNumberB()</td>
<td>431</td>
</tr>
<tr>
<td>9.11.2.6 ShiftLeft()</td>
<td>431</td>
</tr>
<tr>
<td>9.11.2.7 IsNumeric()</td>
<td>432</td>
</tr>
<tr>
<td>9.12 Memory.c</td>
<td>433</td>
</tr>
<tr>
<td>9.12.1 Description</td>
<td>433</td>
</tr>
<tr>
<td>9.12.2 Includes and Data Definitions</td>
<td>433</td>
</tr>
<tr>
<td>9.12.3 Functions</td>
<td>433</td>
</tr>
</tbody>
</table>
9.12.3.1 MemoryCopy() ......................................... 433
9.12.3.2 MemoryEqual() ....................................... 433
9.12.3.3 MemoryCopy2B() ...................................... 434
9.12.3.4 MemoryConcat2B() .................................... 434
9.12.3.5 MemoryEqual2B() .................................... 434
9.12.3.6 MemorySet() .......................................... 435
9.12.3.7 MemoryPad2B() ........................................ 435
9.12.3.8 Uint16ToByteArray() .................................. 435
9.12.3.9 Uint32ToByteArray() .................................. 435
9.12.3.10 Uint64ToByteArray() ................................. 436
9.12.3.11 ByteArrayToUint16() ................................ 436
9.12.3.12 ByteArrayToUint32() ................................ 436
9.12.3.13 ByteArrayToUint64() ................................ 437

9.13 Power.c ..................................................... 438

9.13.1 Description .............................................. 438
9.13.2 Includes and Data Definitions .......................... 438
9.13.3 Functions ................................................ 438

9.13.3.1 TPMInit() ............................................... 438
9.13.3.2 TPMRegisterStartup() ................................. 438
9.13.3.3 TPMIsStarted() ....................................... 438

9.14 PropertyCap.c .............................................. 440

9.14.1 Description .............................................. 440
9.14.2 Includes .................................................. 440
9.14.3 Functions ................................................ 440

9.14.3.1 TPMPropertyIsDefined() ............................. 440
9.14.3.2 TPMCapGetProperty() ................................. 447

9.15 Response.c .................................................. 449

9.15.1 Description .............................................. 449
9.15.2 Includes and Defines .................................... 449
9.15.3 BuildResponseHeader() .................................. 449

9.16 ResponseCodeProcessing.c ................................. 450

9.16.1 Description .............................................. 450
9.16.2 Includes and Defines .................................... 450
9.16.3 RcSafeAddToResult() .................................... 450

9.17 TpmFail.c .................................................... 451

9.17.1 Includes, Defines, and Types ......................... 451
9.17.2 Typedefs .................................................. 451
9.17.3 Local Functions ........................................ 451

9.17.3.1 MarshalUint16() ...................................... 452
9.17.3.2 MarshalUint32() ...................................... 452
9.17.3.3 UnmarshalHeader() ................................... 452

9.17.4 Public Functions ........................................ 453

9.17.4.1 SetForceFailureMode() ............................... 453
9.17.4.2 TpmFail() .............................................. 453
9.17.4.3 TpmFailureMode ....................................... 454
9.17.4.4 UnmarshalFail() ...................................... 457

10 Cryptographic Functions ..................................... 458

10.1 Headers .................................................... 458

10.1.1 BnValues.h ............................................... 458
10.1.2 CryptEcc.h ................................................................. 463
10.1.2.1 Introduction .......................................................... 463
10.1.2.2 Structures .............................................................. 463
10.1.3 CryptHash.h .............................................................. 464
10.1.3.1 Introduction .......................................................... 464
10.1.3.2 Hash-related Structures ........................................... 464
10.1.3.3 HMAC State Structures ............................................ 467
10.1.4 CryptRand.h ............................................................. 468
10.1.4.1 Introduction .......................................................... 468
10.1.4.2 DRBG Structures and Defines .................................... 466
10.1.5 CryptRsa.h ............................................................... 471
10.1.6 CryptTest.h .............................................................. 472
10.1.7 HashTestData.h .......................................................... 473
10.1.8 KdfTestData.h ............................................................ 474
10.1.9 RsaTestData.h ............................................................ 475
10.1.10 SymmetricTestData.h ................................................ 481
10.1.11 SymmetricTest.h ........................................................ 483
10.1.11.1 Introduction ........................................................ 483
10.1.11.2 Symmetric Test Structures ....................................... 483
10.1.12 EccTestData.h .......................................................... 484
10.1.13 CryptSym.h ............................................................. 486
10.1.13.1 Introduction ........................................................ 486
10.1.13.2 Includes, Defines, and Typesdef ................................ 486
10.2 Source ................................................................. 488
10.2.1 AlgorithmTests.c ......................................................... 488
10.2.1.1 Introduction ........................................................ 488
10.2.1.2 Includes and Defines .............................................. 488
10.2.1.3 Hash Tests ............................................................ 488
10.2.1.3.1 Description ....................................................... 488
10.2.1.3.2 TestHash() .......................................................... 488
10.2.1.4 Symmetric Test Functions ......................................... 490
10.2.1.4.1 MakeIV() ............................................................. 490
10.2.1.4.2 TestSymmetricAlgorithm() ..................................... 490
10.2.1.4.3 AllSymAreDone() .................................................. 491
10.2.1.4.4 AllModesAreDone() .............................................. 491
10.2.1.4.5 TestSymmetric() .................................................... 491
10.2.1.5 RSA Tests .............................................................. 492
10.2.1.5.1 Introduction ....................................................... 493
10.2.1.5.2 RsaKeyInitialize() ............................................... 493
10.2.1.5.3 TestRsaEncryptDecrypt() ....................................... 493
10.2.1.5.4 TestRsaSignAndVerify() ........................................ 495
10.2.1.5.5 TestRSA() ............................................................ 496
10.2.1.6 ECC Tests ............................................................. 497
10.2.1.6.1 LoadEccParameter() ............................................... 497
10.2.1.6.2 LoadEccPoint() ...................................................... 497
10.2.1.6.3 TestECDH() .......................................................... 497
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2.1.6.4 TestEccSignAndVerify()</td>
<td>498</td>
</tr>
<tr>
<td>10.2.1.6.5 TestKDFa()</td>
<td>499</td>
</tr>
<tr>
<td>10.2.1.6.6 TestEcc()</td>
<td>499</td>
</tr>
<tr>
<td>10.2.1.6.7 TestAlgorithm()</td>
<td>500</td>
</tr>
<tr>
<td>10.2.2 BnConvert.c</td>
<td>504</td>
</tr>
<tr>
<td>10.2.2.1 Introduction</td>
<td>504</td>
</tr>
<tr>
<td>10.2.2.2 Includes</td>
<td>504</td>
</tr>
<tr>
<td>10.2.2.3 Functions</td>
<td>504</td>
</tr>
<tr>
<td>10.2.2.3.1 BnFromBytes()</td>
<td>504</td>
</tr>
<tr>
<td>10.2.2.3.2 BnFrom2Bt()</td>
<td>505</td>
</tr>
<tr>
<td>10.2.2.3.3 BnFromHex()</td>
<td>505</td>
</tr>
<tr>
<td>10.2.2.3.4 BnToBytes()</td>
<td>506</td>
</tr>
<tr>
<td>10.2.2.3.5 BnTo2Bt()</td>
<td>507</td>
</tr>
<tr>
<td>10.2.2.3.6 BnPointFrom2Bt()</td>
<td>507</td>
</tr>
<tr>
<td>10.2.2.3.7 BnPointTo2Bt()</td>
<td>507</td>
</tr>
<tr>
<td>10.2.3 BnMath.c</td>
<td>509</td>
</tr>
<tr>
<td>10.2.3.1 Introduction</td>
<td>509</td>
</tr>
<tr>
<td>10.2.3.2 Includes</td>
<td>509</td>
</tr>
<tr>
<td>10.2.3.3 Functions</td>
<td>509</td>
</tr>
<tr>
<td>10.2.3.3.1 AddSame()</td>
<td>509</td>
</tr>
<tr>
<td>10.2.3.3.2 CarryProp()</td>
<td>510</td>
</tr>
<tr>
<td>10.2.3.3.3 BnAdd()</td>
<td>510</td>
</tr>
<tr>
<td>10.2.3.3.4 BnAddWord()</td>
<td>511</td>
</tr>
<tr>
<td>10.2.3.3.5 SubSame()</td>
<td>511</td>
</tr>
<tr>
<td>10.2.3.3.6 BorrowProp()</td>
<td>512</td>
</tr>
<tr>
<td>10.2.3.3.7 BnSub()</td>
<td>512</td>
</tr>
<tr>
<td>10.2.3.3.8 BnSubWord()</td>
<td>512</td>
</tr>
<tr>
<td>10.2.3.3.9 BnUnsignedCmp()</td>
<td>513</td>
</tr>
<tr>
<td>10.2.3.3.10 BnUnsignedCmpWord()</td>
<td>513</td>
</tr>
<tr>
<td>10.2.3.3.11 BnModWord()</td>
<td>514</td>
</tr>
<tr>
<td>10.2.3.3.12 Msb()</td>
<td>514</td>
</tr>
<tr>
<td>10.2.3.3.13 BnMsb()</td>
<td>514</td>
</tr>
<tr>
<td>10.2.3.3.14 BnSizeInBits()</td>
<td>515</td>
</tr>
<tr>
<td>10.2.3.3.15 BnSetWord()</td>
<td>515</td>
</tr>
<tr>
<td>10.2.3.3.16 BnSetBit()</td>
<td>515</td>
</tr>
<tr>
<td>10.2.3.3.17 BnTestBit()</td>
<td>516</td>
</tr>
<tr>
<td>10.2.3.3.18 BnMaskBits()</td>
<td>516</td>
</tr>
<tr>
<td>10.2.3.3.19 BnShiftRight()</td>
<td>517</td>
</tr>
<tr>
<td>10.2.3.3.20 BnGenRandomBits()</td>
<td>517</td>
</tr>
<tr>
<td>10.2.3.3.21 BnGenerateRandomInRange()</td>
<td>518</td>
</tr>
<tr>
<td>10.2.4 BnMemory.c</td>
<td>520</td>
</tr>
<tr>
<td>10.2.4.1 Introduction</td>
<td>520</td>
</tr>
<tr>
<td>10.2.4.2 Includes</td>
<td>520</td>
</tr>
<tr>
<td>10.2.4.3 Functions</td>
<td>520</td>
</tr>
<tr>
<td>10.2.4.3.1 BnSetTop()</td>
<td>520</td>
</tr>
<tr>
<td>10.2.4.3.2 BnClearTop()</td>
<td>520</td>
</tr>
<tr>
<td>10.2.4.3.3 BnInitializeWord()</td>
<td>521</td>
</tr>
<tr>
<td>10.2.4.3.4 BnInit()</td>
<td>521</td>
</tr>
<tr>
<td>10.2.4.3.5 BnCopy()</td>
<td>521</td>
</tr>
<tr>
<td>10.2.4.3.6 BnPointCopy()</td>
<td>522</td>
</tr>
<tr>
<td>10.2.4.3.7 BnInitializePoint()</td>
<td>522</td>
</tr>
<tr>
<td>10.2.5 CryptCmac.c</td>
<td>523</td>
</tr>
<tr>
<td>10.2.5.1 Introduction</td>
<td>523</td>
</tr>
</tbody>
</table>
Part 4: Supporting Routines

10.2.6.1 Introduction............................................. 526
10.2.6.2 Includes.................................................. 526
10.2.6.3 Hash/HMAC Functions.................................... 526
  10.2.6.3.1 CryptHmacSign().................................. 526
  10.2.6.3.2 CryptHMACVerifySignature()...................... 526
  10.2.6.3.3 CryptGenerateKeyedHash().......................... 527
  10.2.6.3.4 CryptSchemeAnonymous().......................... 528
10.2.6.4 Symmetric Functions...................................... 528
  10.2.6.4.1 ParmDecryptSym()................................ 528
  10.2.6.4.2 ParmEncryptSym()................................ 529
  10.2.6.4.3 CryptGenerateKeySymmetric()...................... 530
  10.2.6.4.4 CryptXORObfuscation()............................ 531
10.2.6.5 Initialization and shut down............................ 531
  10.2.6.5.1 CryptInit()....................................... 531
  10.2.6.5.2 CryptStartup().................................... 532
10.2.6.6 Algorithm-Independent Functions......................... 533
  10.2.6.6.1 Introduction..................................... 533
  10.2.6.6.2 CryptIsAsymAlgorithm().......................... 533
  10.2.6.6.3 CryptSecretEncrypt().............................. 533
  10.2.6.6.4 CryptSecretDecrypt().............................. 535
  10.2.6.6.5 CryptParameterEncryption()....................... 538
  10.2.6.6.6 CryptParameterDecryption()....................... 539
  10.2.6.6.7 CryptComputeSymmetricUnique()................... 540
  10.2.6.6.8 CryptCreateObject()............................... 541
  10.2.6.6.9 CryptGetSignHashAlg()............................ 543
  10.2.6.6.10 CryptIsSplitSign()................................ 544
  10.2.6.6.11 CryptIsAsymSignScheme()........................ 544
  10.2.6.6.12 CryptIsAsymDecryptScheme()..................... 545
  10.2.6.6.13 CryptSelectSignScheme()......................... 546
  10.2.6.6.14 CryptSign()...................................... 548
  10.2.6.6.15 CryptValidateSignature()........................ 549
  10.2.6.6.16 CryptGetTestResult()............................. 550
  10.2.6.6.17 CryptUniqueSizeValid().......................... 550
  10.2.6.6.18 CryptIsSensitiveSizeValid()..................... 551
  10.2.6.6.19 CryptValidateKeys().............................. 552
  10.2.6.6.20 CryptAlgSetImplemented()....................... 555
  10.2.6.6.21 CryptSelectMac()................................ 555
  10.2.6.6.22 CryptMacIsValidForKey()........................ 556
  10.2.6.6.23 CryptSmacIsValidAlg()........................... 556
  10.2.6.6.24 CryptSymModelsValid()............................ 557

10.2.7 CryptSelfTest.c........................................... 558
  10.2.7.1 Introduction....................................... 558
  10.2.7.2 Functions........................................... 558
  10.2.7.2.1 RunSelfTest().................................... 558
  10.2.7.2.2 CryptSelfTest().................................. 558
  10.2.7.2.3 CryptIncrementalSelfTest()...................... 559
10.2.12.3 Signing Functions ................................................................. 594
  10.2.12.3.1 BnSignEcdsa() .......................................................... 594
  10.2.12.3.2 BnSignEccdaa() ....................................................... 595
  10.2.12.3.3 SchnorrReduce() ...................................................... 597
  10.2.12.3.4 SchnorrEcc() ........................................................ 597
  10.2.12.3.5 BnHexEqual() .......................................................... 598
  10.2.12.3.6 BnSignEcSm2() ........................................................ 599
  10.2.12.3.7 CryptEccSign() ....................................................... 600
  10.2.12.3.8 BnValidateSignatureEcdsa() ...................................... 602
  10.2.12.3.9 BnValidateSignatureEccSm2() .................................... 603
  10.2.12.3.10 BnValidateSignatureEccSchnorr() ............................. 604
  10.2.12.3.11 CryptEccValidateSignature() .................................... 605
  10.2.12.3.12 CryptEccCommitCompute() ........................................ 606

10.2.13 CryptHash.c ........................................................................ 608
  10.2.13.1 Description .................................................................... 608
  10.2.13.2 Includes, Defines, and Types ........................................... 608
  10.2.13.3 Obligatory Initialization Functions .................................... 608
  10.2.13.3.1 CryptHashInit() ........................................................ 608
  10.2.13.3.2 CryptHashStartup() ................................................... 608

10.2.13.4 Hash Information Access Functions ...................................... 609
  10.2.13.4.1 Introduction .............................................................. 609
  10.2.13.4.2 CryptGetHashDef() .................................................... 609
  10.2.13.4.3 CryptHashIsValidAlg() .............................................. 609
  10.2.13.4.4 GetHashInfoPointer() ............................................... 610
  10.2.13.4.5 CryptHashGetAlgByIndex() ....................................... 610
  10.2.13.4.6 CryptHashGetDigestSize() ......................................... 611
  10.2.13.4.7 CryptHashGetBlockSize() .......................................... 611
  10.2.13.4.8 CryptHashGetDer ..................................................... 611
  10.2.13.4.9 CryptHashGetContextAlg() ....................................... 612

10.2.13.5 State Import and Export .................................................... 612
  10.2.13.5.1 CryptHashCopyState ................................................ 612
  10.2.13.5.2 CryptHashExportState() .......................................... 612
  10.2.13.5.3 CryptHashImportState() .......................................... 613

10.2.13.6 State Modification Functions ............................................. 614
  10.2.13.6.1 HashEnd() .............................................................. 614
  10.2.13.6.2 CryptHashStart() .................................................... 614
  10.2.13.6.3 CryptDigestUpdate() ............................................... 615
  10.2.13.6.4 CryptHashEnd() ...................................................... 615
  10.2.13.6.5 CryptHashBlock() .................................................... 616
  10.2.13.6.6 CryptDigestUpdate2B() ............................................ 616
  10.2.13.6.7 CryptHashEnd2B() ................................................... 617
  10.2.13.6.8 CryptDigestUpdateInt() ............................................ 617

10.2.13.7 HMAC Functions ............................................................... 617
  10.2.13.7.1 CryptHmacStart ...................................................... 617
  10.2.13.7.2 CryptHmacEnd() ..................................................... 619
  10.2.13.7.3 CryptHmacStart2B() ............................................... 619
  10.2.13.7.4 CryptHmacEnd2B() ................................................... 620

10.2.13.8 Mask and Key Generation Functions .................................... 620
  10.2.13.8.1 CryptMGF1() .......................................................... 620
  10.2.13.8.2 CryptKDFa() .......................................................... 621
  10.2.13.8.3 CryptKDFe() .......................................................... 623
10.2.14 CryptHashData.c ................................................................. 625
10.2.15 CryptPrime.c ................................................................. 626
10.2.15.1 Introduction ................................................................. 626
10.2.15.2 Functions ................................................................. 626
10.2.15.2.1 Root2() ................................................................. 626
10.2.15.2.2 IsPrimeInt() ............................................................... 626
10.2.15.2.3 BnIsPrime() ............................................................... 627
10.2.15.2.4 MillerRabinRounds() ................................................ 627
10.2.15.2.5 MillerRabin() ............................................................... 628
10.2.15.2.6 RsaCheckPrime() ....................................................... 629
10.2.15.2.7 AdjustPrimeCandidate() .............................................. 630
10.2.15.2.8 BnGeneratePrimeForRSA() ............................................ 630
10.2.16 CryptPrimeSieve.c ............................................................... 632
10.2.16.1 Includes and defines .................................................... 632
10.2.16.2 Functions ................................................................. 632
10.2.16.2.1 RsaAdjustPrimeLimit() ............................................... 632
10.2.16.2.2 RsaNextPrime() .............................................................. 632
10.2.16.2.3 BitsInArray() ............................................................... 634
10.2.16.2.4 FindNthSetBit() ............................................................. 634
10.2.16.2.5 PrimeSieve() ............................................................... 635
10.2.16.2.6 SetFieldSize() ............................................................... 637
10.2.16.2.7 PrimeSelectWithSieve() ............................................... 637
10.2.17 CryptRand.c ................................................................. 641
10.2.17.1 Introduction ................................................................. 641
10.2.17.2 Derivation Functions .................................................... 641
10.2.17.2.1 Description ............................................................... 641
10.2.17.2.2 Derivation Function Defines and Structures .................... 642
10.2.17.2.3 DFCompute() ............................................................... 642
10.2.17.2.4 DFStart() ................................................................. 642
10.2.17.2.5 DFUpdate() ............................................................... 643
10.2.17.2.6 DFDel() ................................................................. 643
10.2.17.2.7 DFIterator() ................................................................. 644
10.2.17.2.8 DRBG_GetEntropy() ..................................................... 644
10.2.17.2.9 Increment() ............................................................... 645
10.2.17.2.10 EncryptDRBG() .......................................................... 645
10.2.17.2.11 DRBG_Update() .......................................................... 646
10.2.17.2.12 DRBG_Reset() ............................................................. 647
10.2.17.2.13 DRBG_SelfTest() ........................................................ 648
10.2.17.3 Public Interface ............................................................ 649
10.2.17.3.1 Description ............................................................... 649
10.2.17.3.2 CryptRandomStir() .................................................... 649
10.2.17.3.3 CryptRandomGenerate() ............................................... 650
10.2.17.3.4 DRBG_Generate() ....................................................... 655
10.2.17.3.5 DRBG_Initialize() ....................................................... 657
10.2.17.3.6 DRBG_Uninitialize() ................................................. 658
10.2.18 CryptRsa.c ................................................................. 659
10.2.18.1 Introduction ............................................................... 659
10.2.18.2 Includes ................................................................. 659
10.2.18.3 Obligatory Initialization Functions .................................. 659
10.2.18.3.1 CryptRsaInit() .............................................................. 659
10.2.18.3.2 CryptRsaStartup() ........................................................ 659
10.2.18.4 Internal Functions .......................................................... 659
  10.2.18.4.1 ComputePrivateExponent() ........................................ 660
  10.2.18.4.2 RsaPrivateKeyOp() ................................................ 661
  10.2.18.4.3 RSAEP() ............................................................. 661
  10.2.18.4.4 RSADP() ............................................................. 662
  10.2.18.4.5 OaepEncode() ..................................................... 663
  10.2.18.4.6 OaepDecode() ..................................................... 664
  10.2.18.4.7 PKCS#1_v1_5Encode() .......................................... 665
  10.2.18.4.8 RSAES_Decode() ................................................ 666
  10.2.18.4.9 PssEncode() ...................................................... 667
  10.2.18.4.10 PssDecode() ..................................................... 668
  10.2.18.4.11 RSASSA_Encode() ............................................. 670
  10.2.18.4.12 RSASSA_Decode() ............................................. 670
10.2.18.5 Externally Accessible Functions ..................................... 671
  10.2.18.5.1 CryptRsaSelectScheme() ....................................... 671
  10.2.18.5.2 CryptRsaLoadPrivateExponent() ................................ 672
  10.2.18.5.3 CryptRsaEncrypt() ............................................. 673
  10.2.18.5.4 CryptRsaDecrypt() ............................................. 674
  10.2.18.5.5 CryptRsaSign() ................................................ 675
  10.2.18.5.6 CryptRsaValidateSignature() ................................... 676
  10.2.18.5.7 CryptRsaGenerateKey() ......................................... 677
10.2.19 CryptSmac.c .................................................................... 680
  10.2.19.1 Introduction ............................................................. 680
  10.2.19.2 Includes, Defines, and Typedefs ................................... 680
    10.2.19.2.1 CryptSmacStart() ............................................. 680
    10.2.19.2.2 CryptMacStart() .............................................. 680
    10.2.19.2.3 CryptMacEnd() ............................................... 681
    10.2.19.2.4 CryptMacEnd2B() ............................................. 681
10.2.20 CryptSym.c .................................................................... 682
  10.2.20.1 Introduction ............................................................. 682
  10.2.20.2 Includes, Defines, and Typedefs ................................... 682
    10.2.20.3 Initialization and Data Access Functions ......................... 682
      10.2.20.3.1 CryptSymInit() ............................................. 682
      10.2.20.3.2 CryptSymStartup() ......................................... 682
      10.2.20.3.3 CryptGetSymmetricBlockSize() ............................. 682
    10.2.20.4 Symmetric Encryption ........................................... 683
      10.2.20.4.1 CryptSymmetricDecrypt() .................................. 686
      10.2.20.4.2 CryptSymKeyValidate() .................................... 689
10.2.21 PrimeData.c ..................................................................... 690
10.2.22 RsaKeyCache.c .................................................................. 696
  10.2.22.1 Introduction ............................................................. 696
  10.2.22.2 Includes, Types, Locals, and Defines .............................. 696
    10.2.22.2.1 RsaKeyCacheControl() ........................................ 697
    10.2.22.2.2 InitializeKeyCache() .......................................... 697
    10.2.22.2.3 KeyCacheLoaded() ............................................ 699
    10.2.22.2.4 GetCachedRsaKey() ............................................ 699
10.2.23 Ticket.c ........................................................................ 701
  10.2.23.1 Introduction ............................................................. 701
  10.2.23.2 Includes .................................................................... 701
  10.2.23.3 Functions ............................................................... 701
Part 4: Supporting Routines

C.3.3. Simulator Functions ................................................................. 747
  C.3.3.1. Introduction ........................................................................ 747
  C.3.3.2. _plat_TimerReset() .......................................................... 747
  C.3.3.3. _plat_TimerRestart() ......................................................... 747
C.3.4. Functions Used by TPM .............................................................. 748
  C.3.4.1. Introduction ........................................................................ 748
  C.3.4.2. _plat_RealTime() ............................................................... 748
  C.3.4.3. _plat_TimerRead() ............................................................. 748
  C.3.4.4. _plat_TimerWasReset() ...................................................... 750
  C.3.4.5. _plat_TimerWasStopped() .................................................. 750
  C.3.4.6. _plat_ClockAdjustRate() .................................................... 750
C.4 Entropy.c ....................................................................................... 752
  C.4.1. Includes and Local Values ....................................................... 752
  C.4.2. Functions .............................................................................. 752
    C.4.2.1. rand32() ........................................................................ 752
    C.4.2.2. _plat_GetEntropy() ......................................................... 753
C.5 LocalityPlat.c ................................................................................ 755
  C.5.1. Includes ................................................................................ 755
  C.5.2. Functions .............................................................................. 755
    C.5.2.1. _plat_LocalityGet() ......................................................... 755
    C.5.2.2. _plat_LocalitySet() ......................................................... 755
C.6 NVMem.c ....................................................................................... 756
  C.6.1. Description ........................................................................... 756
  C.6.2. Includes and Local ................................................................. 756
  C.6.3. Functions .............................................................................. 756
    C.6.3.1. NvFileOpen() ................................................................ 756
    C.6.3.2. NvFileCommit() .............................................................. 756
    C.6.3.3. NvFileSize() .................................................................. 757
    C.6.3.4. _plat_NvErrors() ........................................................... 757
    C.6.3.5. _plat_NVEnable() ........................................................... 758
    C.6.3.6. _plat_NVDisable() .......................................................... 759
    C.6.3.7. _plat_IsNVAvailable() .................................................... 759
    C.6.3.8. _plat_NVMemoryRead() .................................................. 759
    C.6.3.9. _plat_NVIsDifferent() ...................................................... 760
    C.6.3.10. _plat_NVMemoryWrite() ............................................... 760
    C.6.3.11. _plat_NVMemoryClear() ................................................. 760
    C.6.3.12. _plat_NVMemoryMove() ............................................... 761
    C.6.3.13. _plat_NVCommit() ....................................................... 761
    C.6.3.14. _plat_SetNVAvail() ....................................................... 761
    C.6.3.15. _plat_ClearNVAvail() ...................................................... 761
C.7 PowerPlat.c ................................................................................... 763
  C.7.1. Includes and Function Prototypes ........................................... 763
  C.7.2. Functions .............................................................................. 763
    C.7.2.1. _plat_Signal_PowerOn() .................................................. 763
    C.7.2.2. _plat_WasPowerLost() .................................................... 763
    C.7.2.3. _plat_Signal_Reset() ....................................................... 763
    C.7.2.4. _plat_Signal_PowerOff() .................................................. 764
C.8 Platform_fp.h ............................................................................... 765
  C.8.1. From Cancel.c ..................................................................... 765
C.8.2. From Clock.c .......................................................... 765
  C.8.2.1. _plat__TimerReset() .................................. 765
  C.8.2.2. _plat__TimerRestart() .................................. 765
  C.8.2.3. _plat__RealTime() ....................................... 766
  C.8.2.4. _plat__TimerRead() ...................................... 766
  C.8.2.5. _plat__TimerWasReset() ................................ 766
  C.8.2.6. _plat__TimerWasStopped() ............................. 766
  C.8.2.7. _plat__ClockAdjustRate() .............................. 767
C.8.3. From Entropy.c ..................................................... 767
  C.8.3.1. _plat__GetEntropy() ................................... 767
C.8.4. From LocalityPlat.c .............................................. 767
  C.8.4.1. _plat__LocalityGet() .................................... 767
  C.8.4.2. _plat__LocalitySet() .................................... 767
C.8.5. From NVMem.c ....................................................... 767
  C.8.5.1. _plat__NvErrors() ...................................... 767
  C.8.5.2. _plat__NVEnable() ...................................... 768
  C.8.5.3. _plat__NVDisable() ...................................... 768
  C.8.5.4. _plat__IsNvAvailable() ................................ 768
  C.8.5.5. _plat__NvMemoryRead() ................................ 768
  C.8.5.6. _plat__NvIsDifferent() ................................ 769
  C.8.5.7. _plat__NvMemoryWrite() ................................ 769
  C.8.5.8. _plat__NvMemoryClear() ................................ 769
  C.8.5.9. _plat__NvMemoryMove() ................................ 769
  C.8.5.10. _plat__NvCommit() ...................................... 770
  C.8.5.11. _plat__SetNvAvail() ................................... 770
  C.8.5.12. _plat__ClearNvAvail() ................................ 770
C.8.6. From PowerPlat.c .................................................. 770
  C.8.6.1. _plat__Signal_PowerOn() ................................ 770
  C.8.6.2. _plat__WasPowerLost() ................................ 770
  C.8.6.3. _plat__Signal_Reset() .................................. 771
  C.8.6.4. _plat__Signal_PowerOff() ................................ 771
C.8.7. From PPPlat.c ..................................................... 771
  C.8.7.1. _plat__PhysicalPresenceAsserted() .................... 771
  C.8.7.2. _plat__Signal_PhysicalPresenceOn() .................... 771
  C.8.7.3. _plat__Signal_PhysicalPresenceOff() .................. 772
C.8.8. From RunCommand.c ............................................... 772
  C.8.8.1. _plat__RunCommand() .................................... 772
  C.8.8.2. _plat__Fail() ............................................ 772
C.8.9. From Unique.c .................................................... 772
  C.8.10. _plat__GetUnique() ........................................ 772
C.9 PlatformData.h ..................................................... 773
C.10 PlatformData.c ..................................................... 776
  C.10.1. Description ................................................ 776
  C.10.2. Includes .................................................... 776
C.11 PPPlat.c ............................................................. 777
  C.11.1. Description ................................................ 777
C.11.2. Includes ........................................................................................................777
C.11.3. Functions ....................................................................................................777
C.11.3.1. _plat_PhysicalPresenceAsserted() .....................................................777
C.11.3.2. _plat__Signal_PhysicalPresenceOn() .................................................777
C.11.3.3. _plat__Signal_PhysicalPresenceOff() ..................................................777

C.12 RunCommand.c.............................................................................................779
C.12.1. Introduction ..............................................................................................779
C.12.2. Includes and locals ...................................................................................779
C.12.3. Functions ..................................................................................................779
C.12.3.1. _plat__RunCommand() ......................................................................779
C.12.3.2. _plat__Fail() .......................................................................................779

C.13 Unique.c ..........................................................................................................780
C.13.1. Introduction ..............................................................................................780
C.13.2. Includes ....................................................................................................780
C.13.3. _plat__GetUnique() ................................................................................780

Annex D (informative) Remote Procedure Interface ...........................................781
D.1 Introduction ....................................................................................................781
D.2 Simulator_fp.h ...............................................................................................781
D.2.1. From TcpServer.c .....................................................................................781
D.2.1.1. PlatformServer() ................................................................................781
D.2.1.2. PlatformSvcRoutine() ........................................................................781
D.2.1.3. PlatformSignalService() ......................................................................781
D.2.1.4. RegularCommandService() ..................................................................781
D.2.1.5. StartTcpServer() ..................................................................................782
D.2.1.6. ReadBytes() ........................................................................................782
D.2.1.7. WriteBytes() .........................................................................................782
D.2.1.8. WriteUINT32() ....................................................................................782
D.2.1.9. ReadVarBytes() ....................................................................................782
D.2.1.10. WriteVarBytes() ................................................................................783
D.2.1.11. TpmServer() ........................................................................................783

D.2.2. From TPMCmdp.c .......................................................................................783
D.2.2.1. Signal__PowerOn() ..............................................................................783
D.2.2.2. Signal__Restart() ................................................................................783
D.2.2.3. Signal__PowerOff() ............................................................................783
D.2.2.4. __rpc__ForceFailureMode() .................................................................783
D.2.2.5. __rpc__Signal__PhysicalPresenceOn() ................................................784
D.2.2.6. __rpc__Signal__PhysicalPresenceOff() ..............................................784
D.2.2.7. __rpc__Signal__Hash__Start() ...............................................................784
D.2.2.8. __rpc__Signal__Hash__Data() ...............................................................784
D.2.2.9. __rpc__Signal__Hash__End() ...............................................................784
D.2.2.10. __rpc__Send__Command() .................................................................784
D.2.2.11. __rpc__Signal__CancelOn() .................................................................785
D.2.2.12. __rpc__Signal__CancelOff() ...............................................................785
D.2.2.13. __rpc__Signal__NvOn() ....................................................................785
D.2.2.14. __rpc__Signal__NvOff() ....................................................................785
D.2.2.15. __rpc__RsaKeyCacheControl() .........................................................785
D.2.2.16. __rpc__Shutdowm() .........................................................................785

D.2.3. From TPMCmds.c ......................................................................................786
D.2.3.1. main() .................................................................................................786
D.2.3.2. nothing() .............................................................................................786

D.3 TpmTcpProtocol.h ..........................................................................................787
D.3.1. Introduction .............................................................. 787
D.3.2. Typedefs and Defines .............................................. 787
D.3.3. TPM Commands ....................................................... 787
D.3.4. Enumerations and Structures .................................... 787
D.4  TcpServer.c .................................................................. 789
D.4.1. Description .............................................................. 789
D.4.2. Includes, Locals, Defines and Function Prototypes .......... 789
D.4.3. Functions ............................................................... 789
D.4.3.1. CreateSocket() .................................................... 789
D.4.3.2. PlatformServer() .................................................. 790
D.4.3.3. PlatformSvcRoutine() ........................................... 791
D.4.3.4. PlatformSignalService() ....................................... 792
D.4.3.5. RegularCommandService() ..................................... 793
D.4.3.6. StartTcpServer() .................................................. 793
D.4.3.7. ReadBytes() .......................................................... 794
D.4.3.8. WriteBytes() ......................................................... 794
D.4.3.9. WriteUINT32() ..................................................... 794
D.4.3.10. ReadVarBytes() ................................................... 795
D.4.3.11. WriteVarBytes() .................................................. 796
D.4.3.12. TpmServer() ........................................................ 796
D.5  TPMcmdp.c ................................................................ 799
D.5.1. Description .............................................................. 799
D.5.2. Includes and Data Definitions ..................................... 799
D.5.3. Functions ............................................................... 799
D.5.3.1. Signal_PowerOn() .................................................. 799
D.5.3.2. Signal_Restart() .................................................... 800
D.5.3.3. Signal_PowerOff() .................................................. 800
D.5.3.4. _rpc__ForceFailureMcode() .................................... 800
D.5.3.5. _rpc__Signal_PhysicalPresenceOn() ......................... 800
D.5.3.6. _rpc__Signal_PhysicalPresenceOff() ......................... 801
D.5.3.7. _rpc__Signal_Hash_Start() ...................................... 801
D.5.3.8. _rpc__Signal_Hash_Data() ..................................... 801
D.5.3.9. _rpc__Signal_HashEnd() ......................................... 801
D.5.3.10. _rpc__Send_Command() ......................................... 802
D.5.3.11. _rpc__Signal_CancelOn() ...................................... 802
D.5.3.12. _rpc__Signal_CancelOff() ...................................... 802
D.5.3.13. _rpc__Signal_NvOn() ............................................. 803
D.5.3.14. _rpc__Signal_NvOff() ............................................ 803
D.5.3.15. _rpc__RsaKeyCacheControl() ................................. 803
D.5.3.16. _rpc__Shutdown() ................................................. 804
D.6  TPMcmds.c ................................................................ 805
D.6.1. Description .............................................................. 805
D.6.2. Includes, Defines, Data Definitions, and Function Prototypes .................................................. 805
D.6.3. Functions ............................................................... 805
D.6.3.1. Usage() ................................................................ 805
D.6.3.2. main() ................................................................ 805
D.6.3.3. nothing() ............................................................ 806
1 Scope

This part contains C code that describes the algorithms and methods used by the command code in TPM 2.0 Part 3. The code in this document augments TPM 2.0 Part 2 and TPM 2.0 Part 3 to provide a complete description of a TPM, including the supporting framework for the code that performs the command actions.

Any TPM 2.0 Part 4 code may be replaced by code that provides similar results when interfacing to the action code in TPM 2.0 Part 3. The behavior of code in this document that is not included in an annex is normative, as observed at the interfaces with TPM 2.0 Part 3 code. Code in an annex is provided for completeness, that is, to allow a full implementation of the specification from the provided code.

The code in parts 3 and 4 is written to define the behavior of a compliant TPM. In some cases (e.g., firmware update), it is not possible to provide a compliant implementation. In those cases, any implementation provided by the vendor that meets the general description of the function provided in TPM 2.0 Part 3 would be compliant.

The code in parts 3 and 4 is not written to meet any particular level of conformance nor does this specification require that a TPM meet any particular level of conformance.

2 Terms and definitions

For the purposes of this document, the terms and definitions given in TPM 2.0 Part 1 apply.

3 Symbols and abbreviated terms

For the purposes of this document, the symbols and abbreviated terms given in TPM 2.0 Part 1 apply.

4 Automation

TPM 2.0 Part 2 and 3 are constructed so that they can be processed by an automated parser. For example, TPM 2.0 Part 2 can be processed to generate header file contents such as structures, typedefs, and enums. TPM 2.0 Part 3 can be processed to generate command and response marshaling and unmarshaling code.

The automated processor is not provided to the TCG. It was used to generate the Microsoft Visual Studio TPM simulator files. These files are not specification reference code, but rather design examples.

The automation produces TPM_Types.h, a header representing TPM 2.0 Part 2. It also produces, for each major clause of Part 4, a header of the form _fp.h with the function prototypes.

EXAMPLE The header file for SessionProcess.c is SessionProcess_fp.h.

4.1 Configuration Parser

The tables in the TPM 2.0 Part 2 Annexes are constructed so that they can be processed by a program. The program that processes these tables in the TPM 2.0 Part 2 Annexes is called "The TPM 2.0 Part 2 Configuration Parser."

The tables in the TPM 2.0 Part 2 Annexes determine the configuration of a TPM implementation. These tables may be modified by an implementer to describe the algorithms and commands to be executed in by a specific implementation as well as to set implementation limits such as the number of PCR, sizes of buffers, etc.
The TPM 2.0 Part 2 Configuration Parser produces a set of structures and definitions that are used by the TPM 2.0 Part 2 Structure Parser.

4.2 Structure Parser

4.2.1 Introduction

The program that processes the tables in TPM 2.0 Part 2 (other than the table in the annexes) is called "The TPM 2.0 Part 2 Structure Parser."

NOTE

A Perl script was used to parse the tables in TPM 2.0 Part 2 to produce the header files and unmarshaling code in for the reference implementation.

The TPM 2.0 Part 2 Structure Parser takes as input the files produced by the TPM 2.0 Part 2 Configuration Parser and the same TPM 2.0 Part 2 specification that was used as input to the TPM 2.0 Part 2 Configuration Parser. The TPM 2.0 Part 2 Structure Parser will generate all of the C structure constant definitions that are required by the TPM interface. Additionally, the parser will generate unmarshaling code for all structures passed to the TPM, and marshaling code for structures passed from the TPM.

The unmarshaling code produced by the parser uses the prototypes defined below. The unmarshaling code will perform validations of the data to ensure that it is compliant with the limitations on the data imposed by the structure definition and use the response code provided in the table if not.

EXAMPLE: The definition for a TPMI_RH_PROVISION indicates that the primitive data type is a TPM_HANDLE and the only allowed values are TPM_RH_OWNER and TPM_RH_PLATFORM. The definition also indicates that the TPM shall indicate TPM_RC_HANDLE if the input value is not one of these values. The unmarshaling code will validate that the input value has one of those allowed values and return TPM_RC_HANDLE if not.

The sections below describe the function prototypes for the marshaling and unmarshaling code that is automatically generated by the TPM 2.0 Part 2 Structure Parser. These prototypes are described here as the unmarshaling and marshaling of various types occurs in places other than when the command is being parsed or the response is being built. The prototypes and the description of the interface are intended to aid in the comprehension of the code that uses these auto-generated routines.

4.2.2 Unmarshaling Code Prototype

4.2.2.1 Simple Types and Structures

The general form for the unmarshaling code for a simple type or a structure is:

```c
TPM_RC TYPE_Unmarshal(TYPE *target, BYTE **buffer, INT32 *size);
```

Where:

- `TYPE` name of the data type or structure
- `*target` location in the TPM memory into which the data from `**buffer` is placed
- `**buffer` location in input buffer containing the most significant octet (MSO) of `*target`
- `*size` number of octets remaining in `**buffer`

When the data is successfully unmarshaled, the called routine will return TPM_RC_SUCCESS. Otherwise, it will return a Format-One response code (see TPM 2.0 Part 2).
If the data is successfully unmarshaled, *buffer is advanced to the first octet of the next parameter in the input buffer and size is reduced by the number of octets removed from the buffer.

When the data type is a simple type, the parser will generate code that will unmarshal the underlying type and then perform checks on the type as indicated by the type definition.

When the data type is a structure, the parser will generate code that unmarshals each of the structure elements in turn and performs any additional parameter checks as indicated by the data type.

### 4.2.2.2 Union Types

When a union is defined, an extra parameter is defined for the unmarshaling code. This parameter is the selector for the type. The unmarshaling code for the union will unmarshal the type indicated by the selector.

The function prototype for a union has the form:

```
TPM_RC TYPE_Unmarshal(TYPE *target, BYTE **buffer, INT32 *size, UINT32 selector);
```

where:

- **TYPE** name of the union type or structure
- **target** location in the TPM memory into which the data from **buffer** is placed
- **buffer** location in input buffer containing the most significant octet (MSO) of **target**
- **size** number of octets remaining in **buffer**
- **selector** union selector that determines what will be unmarshaled into **target**

### 4.2.2.3 Null Types

In some cases, the structure definition allows an optional "null" value. The "null" value allows the use of the same C type for the entity even though it does not always have the same members.

For example, the TPMI_ALG_HASH data type is used in many places. In some cases, TPM_ALG_NULL is permitted and in some cases it is not. If two different data types had to be defined, the interfaces and code would become more complex because of the number of cast operations that would be necessary. Rather than encumber the code, the "null" value is defined and the unmarshaling code is given a flag to indicate if this instance of the type accepts the "null" parameter or not. When the data type has a "null" value, the function prototype is

```
TPM_RC TYPE_Unmarshal(TYPE *target, BYTE **buffer, INT32 *size, BOOL flag);
```

The parser detects when the type allows a "null" value and will always include flag in any call to unmarshal that type. flag TRUE indicates that null is accepted.

### 4.2.2.4 Arrays

Any data type may be included in an array. The function prototype use to unmarshal an array for a TYPE is

```
TPM_RC TYPE_Array_Unmarshal(TYPE *target, BYTE **buffer, INT32 *size, INT32 count);
```

The generated code for an array uses a count-limited loop within which it calls the unmarshaling code for TYPE.
4.2.3 Marshaling Code Function Prototypes

4.2.3.1 Simple Types and Structures

The general form for the marshaling code for a simple type or a structure is:

```c
UINT16 TYPE_Marshal(TYPE *source, BYTE **buffer, INT32 *size);
```

Where:

- **TYPE**
  - name of the data type or structure
- **source**
  - location in the TPM memory containing the value that is to be marshaled into the designated buffer
- **buffer**
  - location in the output buffer where the first octet of the **TYPE** is to be placed
- **size**
  - number of octets remaining in **buffer**.

If **buffer** is a NULL pointer, then no data is marshaled, but the routine will compute and return the size of the memory required to marshal the indicated type. **size** is not changed.

If **buffer** is not a NULL pointer, data is marshaled, **buffer** is advanced to point to the first octet of the next location in the output buffer, and the called routine will return the number of octets marshaled into **buffer**. This occurs even if **size** is a NULL pointer. If **size** is a not NULL pointer **size** is reduced by the number of octets placed in the buffer.

When the data type is a simple type, the parser will generate code that will marshal the underlying type. The presumption is that the TPM internal structures are consistent and correct so the marshaling code does not validate that the data placed in the buffer has a permissible value. The presumption is also that the **size** is sufficient for the source being marshaled.

When the data type is a structure, the parser will generate code that marshals each of the structure elements in turn.

4.2.3.2 Union Types

An extra parameter is defined for the marshaling function of a union. This parameter is the selector for the type. The marshaling code for the union will marshal the type indicated by the selector.

The function prototype for a union has the form:

```c
UINT16 TYPE_Marshal(TYPE *source, BYTE **buffer, INT32 *size, UINT32 selector);
```

The parameters have a similar meaning as those in 4.2.2.2 but the data movement is from **source** to **buffer**.

4.2.3.3 Arrays

Any type may be included in an array. The function prototype use to unmarshal an array is:

```c
UINT16 TYPE_Array_Marshal(TYPE *source, BYTE **buffer, INT32 *size, INT32 count);
```

The generated code for an array uses a count-limited loop within which it calls the marshaling code for **TYPE**.
4.3 Part 3 Parsing

The Command / Response tables in Part 3 of this specification are processed by scripts to produce the command-specific data structures used by functions in this TPM 2.0 Part 4. They are:

- **CommandAttributeData.h** -- This file contains the command attributes reported by TPM2_GetCapability.
- **CommandAttributes.h** – This file contains the definition of command attributes that are extracted by the parsing code. The file mainly exists to ensure that the parsing code and the function code are using the same attributes.
- **CommandDispatchData.h** – This file contains the data definitions for the table driven version of the command dispatcher.

Part 3 parsing also produces special function prototype files as described in 4.4.

4.4 Function Prototypes

For functions that have entry definitions not defined by Part 3 tables, a script is used to extracts function prototypes from the code. For each .c file that is not in Part 3, a file with the same name is created with a suffix of _fp.h. For example, the function prototypes for Create.c will be placed in a file called Create_fp.h. The _fp.h is added because some files have two types of associated headers, the one containing the function prototypes for the file and another containing definitions that are specific to that file.

In some cases, a function will be replaced by a macro. The macro is defined in the .c file and extracted by the function prototype processor. A special comment tag (“//%”) is used to indicate that the line is to be included in the function prototype file. If the “//%” tag occurs at the start of the line, it is deleted. If it occurs later in the line, it is preserved. Removing the “//%” at the start of the line allows the macro to be placed in the .c file with the tag as a prefix, and then show up in the _fp.h file as the actual macro. This allows the code that includes that function prototype code to use the appropriate macro.

For files that contain the command actions, a special _fp.h file is created from the tables in Part 3. These files contain:

- the definition of the input and output structure of the function;
- definition of command-specific return code modifiers (parameter identifiers); and
- the function prototype for the command action function.

Create_fp.h (shown below) is prototypical of the command _fp.h files.

```c
#define TPM_CC_Create // Command must be defined
#ifndef _CREATE_FP_H_
define _CREATE_FP_H_

Input structure definition

typedef struct {
    TPMI_DH_OBJECT              parentHandle;
    TPM2B_SENSITIVE_CREATE      inSensitive;
    TPM2B_PUBLIC                inPublic;
    TPM2B_DATA                  outsideInfo;
    TPML_PCR_SELECTION          creationPCR;
} Create_In;

Output structure definition

typedef struct {
    TPM2B_PRIVATE               outPrivate;
    TPM2B_PUBLIC                outPublic;
    TPM2B_CREATION_DATA         creationData;
```
TPM2B_DIGEST creationHash;
TPMT_TK_CREATION creationTicket;
}

Create_Out;

Response code modifiers
#define RC_Create_parentHandle (TPM_RC_H + TPM_RC_1)
#define RC_Create_inSensitive (TPM_RC_P + TPM_RC_1)
#define RC_Create_inPublic (TPM_RC_P + TPM_RC_2)
#define RC_Create_outsideInfo (TPM_RC_P + TPM_RC_3)
#define RC_Create_creationPCR (TPM_RC_P + TPM_RC_4)

Function prototype
TPM_RC
void TPM2_Create(
    Create_In *in,
    Create_Out *out
);
#endif // _CREATE_FP_H_
#endif // TPM_CC_Create

4.5 Portability

Where reasonable, the code is written to be portable. There are a few known cases where the code is not portable. Specifically, the handling of bit fields will not always be portable. The bit fields are marshaled and unmarshaled as a simple element of the underlying type. For example, a TPMA_SESSION is defined as a bit field in an octet (BYTE). When sent on the interface a TPMA_SESSION will occupy one octet. When unmarshaled, it is unmarshaled as a UINT8. The ramifications of this are that a TPMA_SESSION will occupy the 0th octet of the structure in which it is placed regardless of the size of the structure.

Many compilers will pad a bit field to some "natural" size for the processor, often 4 octets, meaning that sizeof(TPMA_SESSION) would return 4 rather than 1 (the canonical size of a TPMA_SESSION).

For a little endian machine, padding of bit fields should have little consequence since the 0th octet always contains the 0th bit of the structure no matter how large the structure. However, for a big endian machine, the 0th bit will be in the highest numbered octet. When unmarshaling a TPMA_SESSION, the current unmarshaling code will place the input octet at the 0th octet of the TPMA_SESSION. Since the 0th octet is most significant octet, this has the effect of shifting all the session attribute bits left by 24 places.

As a consequence, someone implementing on a big endian machine should do one of two things:

a) allocate all structures as packed to a byte boundary (this may not be possible if the processor does not handle unaligned accesses); or

b) modify the code that manipulates bit fields that are not defined as being the alignment size of the system.

For many RISC processors, option #2 would be the only choice. This is may not be a terribly daunting task since only two attribute structures are not 32-bits (TPMA_SESSION and TPMA_LOCALITY).
5 Header Files

5.1 Introduction

The files in this section are used to define values that are used in multiple parts of the specification and are not confined to a single module.

5.2 BaseTypes.h

```c
#ifndef __BASETYPES_H
#define __BASETYPES_H

NULL definition

#ifndef NULL
#define NULL (0)
#endif

typedef uint8_t UINT8;
typedef uint8_t BYTE;
typedef int8_t INT8;
typedef int BOOL;
typedef uint16_t UINT16;
typedef int16_t INT16;
typedef uint32_t UINT32;
typedef int32_t INT32;
typedef uint64_t UINT64;
typedef int64_t INT64;
#endif // __BASETYPES_H
```
5.3 Capabilities.h

This file contains defines for the number of capability values that will fit into the largest data buffer.

These defines are used in various function in the "support" and the "subsystem" code groups. A module that supports a type that is returned by a capability will have a function that returns the capabilities of the type.

EXAMPLE

PCR.c contains PCRCapGetHandles() and PCRCapGetProperties().

```
1 #ifndef __CAPABILITIES_H
2 #define __CAPABILITIES_H
3 #define MAX_CAP_DATA (MAX_CAP_BUFFER - sizeof(TPM_CAP)-sizeof(UINT32))
4 #define MAX_CAP_ALGS (MAX_CAP_DATA / sizeof(TPMS_ALG_PROPERTY))
5 #define MAX_CAP_HANDLES (MAX_CAP_DATA / sizeof(TPM_HANDLE))
6 #define MAX_CAP_CC (MAX_CAP_DATA / sizeof(TPM_CC))
7 #define MAX_TPM_PROPERTIES (MAX_CAP_DATA / sizeof(TPMS_TAGGED_PROPERTY))
8 #define MAX_PCR_PROPERTIES (MAX_CAP_DATA / sizeof(TPMS_TAGGED_PCR_SELECT))
9 #define MAX_ECC_CURVES (MAX_CAP_DATA / sizeof(TPM_ECC_CURVE))
10 #define MAX_TAGGED_POLICIES (MAX_CAP_DATA / sizeof(TPMS_TAGGED_POLICY))
11 #define MAX_AC_CAPABILITIES (MAX_CAP_DATA / sizeof(TPMS_AC_OUTPUT))
12 #endif
```
5.4 CommandAttributeData.h

This file should only be included by CommandCodeAttributes.c

```c
#ifdef COMMAND_CODE_ATTRIBUTES_
#include "CommandAttributes.h"
#endif

#define PAD_LIST 0
#endif

#define PAD_LIST 1
#endif

This is the command code attribute array for GetCapability(). Both this array and s_commandAttributes provides command code attributes, but tuned for different purpose.

```c
const TPMA_CC s_ccAttr [] = {
    #if (PAD_LIST || CC_NV_UndefineSpaceSpecial)
        TPMA_CC_INITIALIZER(0x011F, 0, 1, 0, 0, 2, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_EvictControl)
        TPMA_CC_INITIALIZER(0x0120, 0, 1, 0, 0, 2, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_HierarchyControl)
        TPMA_CC_INITIALIZER(0x0121, 0, 1, 1, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_NV_UndefineSpace)
        TPMA_CC_INITIALIZER(0x0122, 0, 1, 0, 0, 2, 0, 0, 0),
    #endif
    #if (PAD_LIST )
        TPMA_CC_INITIALIZER(0x0123, 0, 0, 0, 0, 0, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_ChangeEPS)
        TPMA_CC_INITIALIZER(0x0124, 0, 1, 1, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_ChangePPS)
        TPMA_CC_INITIALIZER(0x0125, 0, 1, 1, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_Clear)
        TPMA_CC_INITIALIZER(0x0126, 0, 1, 0, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_HierarchyChangeAuth)
        TPMA_CC_INITIALIZER(0x0129, 0, 1, 0, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_NV_DefineSpace)
        TPMA_CC_INITIALIZER(0x012A, 0, 1, 0, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_PCR_Allocate)
        TPMA_CC_INITIALIZER(0x012B, 0, 1, 0, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_PCR_SetAuthPolicy)
        TPMA_CC_INITIALIZER(0x012C, 0, 1, 0, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_PP_Commands)
        TPMA_CC_INITIALIZER(0x012D, 0, 1, 0, 0, 1, 0, 0, 0),
    #endif
    #if (PAD_LIST || CC_SetPrimaryPolicy)
        TPMA_CC_INITIALIZER(0x012E, 0, 1, 0, 0, 1, 0, 0, 0),
    #endif
```

```
#if (PAD_LIST || CC_FieldUpgradeStart)
TPMA_CC_INITIALIZER(0x012F, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_ClockRateAdjust)
TPMA_CC_INITIALIZER(0x0130, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_CreatePrimary)
TPMA_CC_INITIALIZER(0x0131, 0, 0, 0, 1, 1, 0, 0),
#endif

#if (PAD_LIST || CC_NV_GlobalWriteLock)
TPMA_CC_INITIALIZER(0x0132, 0, 1, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_GetCommandAuditDigest)
TPMA_CC_INITIALIZER(0x0133, 0, 1, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_NV_Increment)
TPMA_CC_INITIALIZER(0x0134, 0, 1, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_NV_SetBits)
TPMA_CC_INITIALIZER(0x0135, 0, 1, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_NV_Extend)
TPMA_CC_INITIALIZER(0x0136, 0, 1, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_NV_Write)
TPMA_CC_INITIALIZER(0x0137, 0, 1, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_NV_WriteLock)
TPMA_CC_INITIALIZER(0x0138, 0, 1, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_DictionaryAttackLockReset)
TPMA_CC_INITIALIZER(0x0139, 0, 1, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_DictionaryAttackParameters)
TPMA_CC_INITIALIZER(0x013A, 0, 1, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_NV_ChangeAuth)
TPMA_CC_INITIALIZER(0x013B, 0, 1, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PCR_Event)
TPMA_CC_INITIALIZER(0x013C, 0, 1, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PCR_Reset)
TPMA_CC_INITIALIZER(0x013D, 0, 1, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_SequenceComplete)
TPMA_CC_INITIALIZER(0x013E, 0, 1, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_SetAlgorithmSet)
TPMA_CC_INITIALIZER(0x013F, 0, 1, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_SetCommandCodeAuditStatus)
TPMA_CC_INITIALIZER(0x0140, 0, 1, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_FieldUpgradeData)
TPMA_CC_INITIALIZER(0x0141, 0, 1, 0, 0, 0, 0, 0),
#endif

#if (PAD_LIST || CC_IncrementalSelfTest)
TPMA_CC_INITIALIZER(0x0142, 0, 1, 0, 0, 0, 0, 0),
#endif

#if (PAD_LIST || CC_SelfTest)
TPMA_CC_INITIALIZER(0x0143, 0, 1, 0, 0, 0, 0, 0),
#endif

#if (PAD_LIST || CC_Startup)
TPMA_CC_INITIALIZER(0x0144, 0, 1, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || CC_Shutdown)
  TPMA_CC_INITIALIZER(0x0145, 0, 1, 0, 0, 0, 0, 0, 0),
#endif

#if (PAD_LIST || CC_StirRandom)
  TPMA_CC_INITIALIZER(0x0146, 0, 1, 0, 0, 0, 0, 0, 0),
#endif

#if (PAD_LIST || CC_ActivateCredential)
  TPMA_CC_INITIALIZER(0x0147, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_Certify)
  TPMA_CC_INITIALIZER(0x0148, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyNV)
  TPMA_CC_INITIALIZER(0x0149, 0, 0, 0, 0, 3, 0, 0, 0),
#endif

#if (PAD_LIST || CC_CertifyCreation)
  TPMA_CC_INITIALIZER(0x0150, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_Duplicate)
  TPMA_CC_INITIALIZER(0x0151, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_GetTime)
  TPMA_CC_INITIALIZER(0x0152, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_GetSessionAuditDigest)
  TPMA_CC_INITIALIZER(0x0153, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_NV_Read)
  TPMA_CC_INITIALIZER(0x0154, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_NV_ReadLock)
  TPMA_CC_INITIALIZER(0x0155, 0, 1, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_ObjectChangeAuth)
  TPMA_CC_INITIALIZER(0x0156, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PolicySecret)
  TPMA_CC_INITIALIZER(0x0157, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_Rewrap)
  TPMA_CC_INITIALIZER(0x0158, 0, 0, 0, 0, 2, 0, 0, 0),
#endif

#if (PAD_LIST || CC_Create)
  TPMA_CC_INITIALIZER(0x0159, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_ECDH_ZGen)
  TPMA_CC_INITIALIZER(0x015A, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || (CC_HMAC || CC_MAC))
  TPMA_CC_INITIALIZER(0x015B, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC.Import)
  TPMA_CC_INITIALIZER(0x015C, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_Load)
  TPMA_CC_INITIALIZER(0x015D, 0, 0, 0, 0, 1, 1, 0, 0),
#endif

#if (PAD_LIST || CC_Quote)
  TPMA_CC_INITIALIZER(0x015E, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_RSA_Decrypt)
  TPMA_CC_INITIALIZER(0x015F, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST)
  TPMA_CC_INITIALIZER(0x0160, 0, 0, 0, 0, 0, 0, 0, 0),
#endif
#if (PAD_LIST || (CC_HMAC_Start || CC_MAC_Start))
TPMA_CC_INITIALIZER(0x015B, 0, 0, 0, 1, 1, 0, 0),
#endif

#if (PAD_LIST || CC_SequenceUpdate)
TPMA_CC_INITIALIZER(0x015C, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_Sign)
TPMA_CC_INITIALIZER(0x015D, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_Unseal)
TPMA_CC_INITIALIZER(0x015E, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicySigned)
TPMA_CC_INITIALIZER(0x0160, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_ContextLoad)
TPMA_CC_INITIALIZER(0x0161, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_ContextSave)
TPMA_CC_INITIALIZER(0x0162, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_ECDH_KeyGen)
TPMA_CC_INITIALIZER(0x0163, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_EncryptDecrypt)
TPMA_CC_INITIALIZER(0x0164, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_FlushContext)
TPMA_CC_INITIALIZER(0x0165, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_LoadExternal)
TPMA_CC_INITIALIZER(0x0166, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_LoadExternal)
TPMA_CC_INITIALIZER(0x0167, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_MakeCredential)
TPMA_CC_INITIALIZER(0x0168, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_NV_ReadPublic)
TPMA_CC_INITIALIZER(0x0169, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyAuthorize)
TPMA_CC_INITIALIZER(0x016A, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyAuthValue)
TPMA_CC_INITIALIZER(0x016B, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyCommandCode)
TPMA_CC_INITIALIZER(0x016C, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyCounterTimer)
TPMA_CC_INITIALIZER(0x016D, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyCpHash)
TPMA_CC_INITIALIZER(0x016E, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyLocality)
TPMA_CC_INITIALIZER(0x016F, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyNameHash)
TPMA_CC_INITIALIZER(0x0170, 0, 0, 0, 1, 0, 0),
#endif
255  #if (PAD_LIST || CC_PolicyOR)
256   TPMA_CC_INITIALIZER(0x0171, 0, 0, 0, 1, 0, 0, 0),
257  #endif
258  #if (PAD_LIST || CC_PolicyTicket)
259   TPMA_CC_INITIALIZER(0x0172, 0, 0, 0, 1, 0, 0, 0),
260  #endif
261  #if (PAD_LIST || CC_ReadPublic)
262   TPMA_CC_INITIALIZER(0x0173, 0, 0, 0, 1, 0, 0, 0),
263  #endif
264  #if (PAD_LIST || CC_RSA_Encrypt)
265   TPMA_CC_INITIALIZER(0x0174, 0, 0, 0, 1, 0, 0, 0),
266  #endif
267  #if (PAD_LIST)
268   TPMA_CC_INITIALIZER(0x0175, 0, 0, 0, 0, 0, 0, 0, 0),
269  #endif
270  #if (PAD_LIST || CC_StartAuthSession)
271   TPMA_CC_INITIALIZER(0x0176, 0, 0, 0, 2, 1, 0, 0, 0),
272  #endif
273  #if (PAD_LIST || CC_VerifySignature)
274   TPMA_CC_INITIALIZER(0x0177, 0, 0, 0, 1, 0, 0, 0, 0),
275  #endif
276  #if (PAD_LIST || CC_ECC_Parameters)
277   TPMA_CC_INITIALIZER(0x0178, 0, 0, 0, 0, 0, 0, 0, 0),
278  #endif
279  #if (PAD_LIST || CC_FirmwareRead)
280   TPMA_CC_INITIALIZER(0x0179, 0, 0, 0, 0, 0, 0, 0, 0),
281  #endif
282  #if (PAD_LIST || CC_GetCapability)
283   TPMA_CC_INITIALIZER(0x017A, 0, 0, 0, 0, 0, 0, 0, 0),
284  #endif
285  #if (PAD_LIST || CC_GetRandom)
286   TPMA_CC_INITIALIZER(0x017B, 0, 0, 0, 0, 0, 0, 0, 0),
287  #endif
288  #if (PAD_LIST || CC_GetTestResult)
289   TPMA_CC_INITIALIZER(0x017C, 0, 0, 0, 0, 0, 0, 0, 0),
290  #endif
291  #if (PAD_LIST || CC_Hash)
292   TPMA_CC_INITIALIZER(0x017D, 0, 0, 0, 0, 0, 0, 0, 0),
293  #endif
294  #if (PAD_LIST || CC_PCR_Read)
295   TPMA_CC_INITIALIZER(0x017E, 0, 0, 0, 0, 0, 0, 0, 0),
296  #endif
297  #if (PAD_LIST || CC_PolicyPCR)
298   TPMA_CC_INITIALIZER(0x017F, 0, 0, 0, 1, 0, 0, 0, 0),
299  #endif
300  #if (PAD_LIST || CC_PolicyRestart)
301   TPMA_CC_INITIALIZER(0x0180, 0, 0, 0, 1, 0, 0, 0, 0),
302  #endif
303  #if (PAD_LIST || CC_ReadClock)
304   TPMA_CC_INITIALIZER(0x0181, 0, 0, 0, 0, 0, 0, 0, 0),
305  #endif
306  #if (PAD_LIST || CC_PCR_Extend)
307   TPMA_CC_INITIALIZER(0x0182, 0, 1, 0, 1, 0, 0, 0, 0),
308  #endif
309  #if (PAD_LIST || CC_PCR_SetAuthValue)
310   TPMA_CC_INITIALIZER(0x0183, 0, 0, 0, 1, 0, 0, 0, 0),
311  #endif
312  #if (PAD_LIST || CC_NV_Certify)
313   TPMA_CC_INITIALIZER(0x0184, 0, 0, 0, 3, 0, 0, 0, 0),
314  #endif
315  #if (PAD_LIST || CC_EventSequenceComplete)
316   TPMA_CC_INITIALIZER(0x0185, 0, 1, 0, 1, 2, 0, 0, 0),
317  #endif
318  #if (PAD_LIST || CC_HashSequenceStart)
319   TPMA_CC_INITIALIZER(0x0186, 0, 0, 0, 0, 1, 0, 0, 0),
320  #endif
#if (PAD_LIST || CC_PolicyPhysicalPresence)
  TPMA_CC_INITIALIZER(0x0187, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyDuplicationSelect)
  TPMA_CC_INITIALIZER(0x0188, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyGetDigest)
  TPMA_CC_INITIALIZER(0x0189, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_TestParms)
  TPMA_CC_INITIALIZER(0x018A, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_Commit)
  TPMA_CC_INITIALIZER(0x018B, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyPassword)
  TPMA_CC_INITIALIZER(0x018C, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_ZGen_2Phase)
  TPMA_CC_INITIALIZER(0x018D, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_EC_Ephemeral)
  TPMA_CC_INITIALIZER(0x018E, 0, 0, 0, 0, 0, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyNvWritten)
  TPMA_CC_INITIALIZER(0x018F, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyTemplate)
  TPMA_CC_INITIALIZER(0x0190, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_CreateLoaded)
  TPMA_CC_INITIALIZER(0x0191, 0, 0, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyAuthorizeNV)
  TPMA_CC_INITIALIZER(0x0192, 0, 0, 0, 0, 3, 0, 0, 0),
#endif

#if (PAD_LIST || CC_EncryptDecrypt2)
  TPMA_CC_INITIALIZER(0x0193, 0, 0, 0, 0, 1, 0, 0, 0),
#endif

#if (PAD_LIST || CC_AC_GetCapability)
  TPMA_CC_INITIALIZER(0x0194, 0, 0, 0, 0, 0, 0, 3, 0, 0),
#endif

#if (PAD_LIST || CC_AC_Send)
  TPMA_CC_INITIALIZER(0x0195, 0, 0, 0, 0, 0, 0, 3, 0, 0),
#endif

#if (PAD_LIST || CC_PolicyAC_SendSelect)
  TPMA_CC_INITIALIZER(0x0196, 0, 0, 0, 0, 0, 0, 1, 0, 0),
#endif

#if (PAD_LIST || CC_Vendor_TCG_Test)
  TPMA_CC_INITIALIZER(0x0000, 0, 0, 0, 0, 0, 0, 0, 1, 0),
#endif

{0}

This is the command code attribute structure.

const COMMAND_ATTRIBUTES s_commandAttributes [] = {
  #if (PAD_LIST || CC_NV_UndefineSpaceSpecial)
    (COMMAND_ATTRIBUTES)(CC_NV_UndefineSpaceSpecial * // 0x011F
     (IS_IMPLEMENTED+HANDLE_1_ADMIN+HANDLE_2_USER+PP_COMMAND)),
  #endif

  #if (PAD_LIST || CC_EvictControl)
    (COMMAND_ATTRIBUTES)(CC_EvictControl * // 0x0120
     (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
  #endif

};
#if (PAD_LIST || CC_HierarchyControl)
  (COMMAND_ATTRIBUTES)(CC_HierarchyControl
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_NV.UndefineSpace)
  (COMMAND_ATTRIBUTES)(CC_NV.UndefineSpace
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST)
  (COMMAND_ATTRIBUTES)(0),
#endif

#if (PAD_LIST || CC_CHANGE_EPS)
  (COMMAND_ATTRIBUTES)(CC_CHANGE_EPS
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_CHANGE_PPS)
  (COMMAND_ATTRIBUTES)(CC_CHANGE_PPS
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_CLEAR)
  (COMMAND_ATTRIBUTES)(CC_CLEAR
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_CLEAR_CONTROL)
  (COMMAND_ATTRIBUTES)(CC_CLEAR_CONTROL
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_CLOCK_SET)
  (COMMAND_ATTRIBUTES)(CC_CLOCK_SET
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_HIERARCHY_CHANGE_AUTH)
  (COMMAND_ATTRIBUTES)(CC_HIERARCHY_CHANGE_AUTH
   (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_NV_DEFINE_SPACE)
  (COMMAND_ATTRIBUTES)(CC_NV_DEFINE_SPACE
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_PCR_ALLOCATE)
  (COMMAND_ATTRIBUTES)(CC_PCR_ALLOCATE
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_PCR_SET_AUTH_POLICY)
  (COMMAND_ATTRIBUTES)(CC_PCR_SET_AUTH_POLICY
   (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_PP_COMMANDS)
  (COMMAND_ATTRIBUTES)(CC_PP_COMMANDS
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_REQUIRED)),
#endif

#if (PAD_LIST || CC_SET_PRIMARY_POLICY)
  (COMMAND_ATTRIBUTES)(CC_SET_PRIMARY_POLICY
   (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_FIELD_UPGRADE_START)
  (COMMAND_ATTRIBUTES)(CC_FIELD_UPGRADE_START
   (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_CLOCK_RATE_ADJUST)
  (COMMAND_ATTRIBUTES)(CC_CLOCK_RATE_ADJUST
   (IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_CREATE_PRIMARY)
  (COMMAND_ATTRIBUTES)(CC_CREATE_PRIMARY
   (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND+ENCRYPT_2+R_HANDLE)),
#endif
#endif

#if (PAD_LIST || CC_NV_GlobalWriteLock)
(COMMAND_ATTRIBUTES)(CC_NV_GlobalWriteLock * // 0x0132
(IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_GetCommandAuditDigest)
(COMMAND_ATTRIBUTES)(CC_GetCommandAuditDigest * // 0x0133
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER+ENCRYPT_2)),
#endif

#if (PAD_LIST || CC_NV_Increment)
(COMMAND_ATTRIBUTES)(CC_NV_Increment * // 0x0134
(IS_IMPLEMENTED+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_NV_SetBits)
(COMMAND_ATTRIBUTES)(CC_NV_SetBits * // 0x0135
(IS_IMPLEMENTED+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_NV_Extend)
(COMMAND_ATTRIBUTES)(CC_NV_Extend * // 0x0136
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_NV_Write)
(COMMAND_ATTRIBUTES)(CC_NV_Write * // 0x0137
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_NV_WriteLock)
(COMMAND_ATTRIBUTES)(CC_NV_WriteLock * // 0x0138
(IS_IMPLEMENTED+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_DictionaryAttackLockReset)
(COMMAND_ATTRIBUTES)(CC_DictionaryAttackLockReset * // 0x0139
(IS_IMPLEMENTED+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_DictionaryAttackParameters)
(COMMAND_ATTRIBUTES)(CC_DictionaryAttackParameters * // 0x013A
(IS_IMPLEMENTED+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_NV_ChangeAuth)
(COMMAND_ATTRIBUTES)(CC_NV_ChangeAuth * // 0x013B
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN)),
#endif

#if (PAD_LIST || CC_PCR_Event)
(COMMAND_ATTRIBUTES)(CC_PCR_Event * // 0x013C
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_PCR_Reset)
(COMMAND_ATTRIBUTES)(CC_PCR_Reset * // 0x013D
(IS_IMPLEMENTED+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_SequenceComplete)
(COMMAND_ATTRIBUTES)(CC_SequenceComplete * // 0x013E
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif

#if (PAD_LIST || CC_SetAlgorithmSet)
(COMMAND_ATTRIBUTES)(CC_SetAlgorithmSet * // 0x013F
(IS_IMPLEMENTED+HANDLE_1_USER)),
#endif

#if (PAD_LIST || CC_SetCommandCodeAuditStatus)
(COMMAND_ATTRIBUTES)(CC_SetCommandCodeAuditStatus * // 0x0140
(IS_IMPLEMENTED+HANDLE_1_USER+PP_COMMAND)),
#endif

#if (PAD_LIST || CC_FieldUpgradeData)
(COMMAND_ATTRIBUTES)(CC_FieldUpgradeData * // 0x0141
(IS_IMPLEMENTED+DECRYPT_2)),
#endif

#if (PAD_LIST || CC_IncrementalSelfTest)
(COMMAND_ATTRIBUTES)(CC_IncrementalSelfTest * // 0x0142
(IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_SelfTest)
(COMMAND_ATTRIBUTES)(CC_SelfTest *
(IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_Startup)
(COMMAND_ATTRIBUTES)(CC_Startup
(IS_IMPLEMENTED+NO_SESSIONS)),
#endif
#if (PAD_LIST || CC_Shutdown)
(COMMAND_ATTRIBUTES)(CC_Shutdown
(IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_StirRandom)
(COMMAND_ATTRIBUTES)(CC_StirRandom *
(IS_IMPLEMENTED+DECRYPT_2)),
#endif
#if (PAD_LIST || CC_ActivateCredential)
(COMMAND_ATTRIBUTES)(CC_ActivateCredential *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Certify)
(COMMAND_ATTRIBUTES)(CC_Certify *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicyNV)
(COMMAND_ATTRIBUTES)(CC_PolicyNV *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_CertifyCreation)
(COMMAND_ATTRIBUTES)(CC_CertifyCreation *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Duplicate)
(COMMAND_ATTRIBUTES)(CC_Duplicate *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_DUP+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_GetTime)
(COMMAND_ATTRIBUTES)(CC_GetTime *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_GetSessionAuditDigest)
(COMMAND_ATTRIBUTES)(CC_GetSessionAuditDigest *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_2_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_NV_Read)
(COMMAND_ATTRIBUTES)(CC_NV_Read *
(IS_IMPLEMENTED+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_NV_ReadLock)
(COMMAND_ATTRIBUTES)(CC_NV_ReadLock *
(IS_IMPLEMENTED+HANDLE_1_USER)),
#endif
#if (PAD_LIST || CC_ObjectChangeAuth)
(COMMAND_ATTRIBUTES)(CC_ObjectChangeAuth *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_ADMIN+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicySecret)
(COMMAND_ATTRIBUTES)(CC_PolicySecret *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ALLOW_TRIAL+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_Rewrap)
(COMMAND_ATTRIBUTES)(CC_Rewrap *
(IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#endif

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"

#define PAD_LIST "1"

#define TRUSTED_LEAVE "1"
#define TRUSTED_LEAVE "1"
#endif

#if (PAD_LIST || CC_EncryptDecrypt)
  (COMMAND_ATTRIBUTES)(CC_EncryptDecrypt
   * // 0x0164
   (IS_IMPLEMENTED+HANDLE_1_USER+ENCRYPT_2)),
#endif

#if (PAD_LIST || CC_FlushContext)
  (COMMAND_ATTRIBUTES)(CC_FlushContext
   * // 0x0165
   (IS_IMPLEMENTED+NO_SESSIONS)),
#endif

#if (PAD_LIST )
  (COMMAND_ATTRIBUTES)(0), // 0x0166
#endif

#if (PAD_LIST || CC_LoadExternal)
  (COMMAND_ATTRIBUTES)(CC_LoadExternal
   * // 0x0167
   (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2+R_HANDLE)),
#endif

#if (PAD_LIST || CC_MakeCredential)
  (COMMAND_ATTRIBUTES)(CC_MakeCredential
   * // 0x0168
   (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2)),
#endif

#if (PAD_LIST || CC_NV_ReadPublic)
  (COMMAND_ATTRIBUTES)(CC_NV_ReadPublic
   * // 0x0169
   (IS_IMPLEMENTED+ENCRYPT_2)),
#endif

#if (PAD_LIST || CC_PolicyAuthorize)
  (COMMAND_ATTRIBUTES)(CC_PolicyAuthorize
   * // 0x016A
   (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_PolicyAuthValue)
  (COMMAND_ATTRIBUTES)(CC_PolicyAuthValue
   * // 0x016B
   (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_PolicyCommandCode)
  (COMMAND_ATTRIBUTES)(CC_PolicyCommandCode
   * // 0x016C
   (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_PolicyCounterTimer)
  (COMMAND_ATTRIBUTES)(CC_PolicyCounterTimer
   * // 0x016D
   (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_PolicyCpHash)
  (COMMAND_ATTRIBUTES)(CC_PolicyCpHash
   * // 0x016E
   (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_PolicyLocality)
  (COMMAND_ATTRIBUTES)(CC_PolicyLocality
   * // 0x016F
   (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_PolicyNameHash)
  (COMMAND_ATTRIBUTES)(CC_PolicyNameHash
   * // 0x0170
   (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_PolicyOR)
  (COMMAND_ATTRIBUTES)(CC_PolicyOR
   * // 0x0171
   (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_PolicyTicket)
  (COMMAND_ATTRIBUTES)(CC_PolicyTicket
   * // 0x0172
   (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_ReadPublic)
  (COMMAND_ATTRIBUTES)(CC_ReadPublic
   * // 0x0173
   (IS_IMPLEMENTED+ENCRYPT_2)),
#endif

#if (PAD_LIST || CC_RSA_Encrypt)
  (COMMAND_ATTRIBUTES)(CC_RSA_Encrypt
   * // 0x0174
713 (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2)),
714 #endif
715 #if (PAD_LIST )
716 (COMMAND_ATTRIBUTES)(0),       // 0x0175
717 #endif
718 #if (PAD_LIST || CC_StartAuthSession)
719 (COMMAND_ATTRIBUTES)(CC_StartAuthSession *  // 0x0176
720 (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2+R_HANDLE)),
721 #endif
722 #if (PAD_LIST || CC_VerifySignature)
723 (COMMAND_ATTRIBUTES)(CC_VerifySignature *  // 0x0177
724 (IS_IMPLEMENTED+DECRYPT_2)),
725 #endif
726 #if (PAD_LIST || CC_ECC_Parameters)
727 (COMMAND_ATTRIBUTES)(CC_ECC_Parameters * // 0x0178
728 (IS_IMPLEMENTED)),
729 #endif
730 #if (PAD_LIST || CC_FirmwareRead)
731 (COMMAND_ATTRIBUTES)(CC_FirmwareRead *  // 0x0179
732 (IS_IMPLEMENTED+ENCRYPT_2)),
733 #endif
734 #if (PAD_LIST || CC_GetCapability)
735 (COMMAND_ATTRIBUTES)(CC_GetCapability *  // 0x017A
736 (IS_IMPLEMENTED)),
737 #endif
738 #if (PAD_LIST || CC_GetRandom)
739 (COMMAND_ATTRIBUTES)(CC_GetRandom * // 0x017B
740 (ISIMPLEMENTED+ENCRYPT_2)),
741 #endif
742 #if (PAD_LIST || CC_GetTestResult)
743 (COMMAND_ATTRIBUTES)(CC_GetTestResult * // 0x017C
744 (IS_IMPLEMENTED+ENCRYPT_2)),
745 #endif
746 #if (PAD_LIST || CC_Hash)
747 (COMMAND_ATTRIBUTES)(CC_Hash * // 0x017D
748 (IS_IMPLEMENTED+ENCRYPT_2)),
749 #endif
750 #if (PAD_LIST || CC_PCR_Read)
751 (COMMAND_ATTRIBUTES)(CC_PCR_Read *  // 0x017E
752 (IS_IMPLEMENTED)),
753 #endif
754 #if (PAD_LIST || CC_PolicyPCR)
755 (COMMAND_ATTRIBUTES)(CC_PolicyPCR * // 0x017F
756 (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
757 #endif
758 #if (PAD_LIST || CC_PolicyRestart)
759 (COMMAND_ATTRIBUTES)(CC_PolicyRestart * // 0x0180
760 (IS_IMPLEMENTED+ALLOW_TRIAL)),
761 #endif
762 #if (PAD_LIST || CC_ReadClock)
763 (COMMAND_ATTRIBUTES)(CC_ReadClock * // 0x0181
764 (IS_IMPLEMENTED)),
765 #endif
766 #if (PAD_LIST || CC_PCR_Extend)
767 (COMMAND_ATTRIBUTES)(CC_PCR_Extend * // 0x0182
768 (IS_IMPLEMENTED+HANDLE_1_USER)),
769 #endif
770 #if (PAD_LIST || CC_PCR_SetAuthValue)
771 (COMMAND_ATTRIBUTES)(CC_PCR_SetAuthValue * // 0x0183
772 (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER)),
773 #endif
774 #if (PAD_LIST || CC_NV_Certify)
775 (COMMAND_ATTRIBUTES)(CC_NV_Certify * // 0x0184
776 (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER+ENCRYPT_2)),
777 #endif
778 #if (PAD_LIST || CC_EventSequenceComplete)
(COMMAND_ATTRIBUTES)(CC_EventSequenceComplete * // 0x0185
  (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+HANDLE_2_USER)),
#endif
#if (PAD_LIST || CC_PolicyPhysicalPresence)
  (COMMAND_ATTRIBUTES)(CC_PolicyPhysicalPresence * // 0x0187
  (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyDuplicationSelect)
  (COMMAND_ATTRIBUTES)(CC_PolicyDuplicationSelect * // 0x0188
  (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyGetDigest)
  (COMMAND_ATTRIBUTES)(CC_PolicyGetDigest * // 0x0189
  (IS_IMPLEMENTED+ALLOW_TRIAL+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_TestParms)
  (COMMAND_ATTRIBUTES)(CC_TestParms * // 0x018A
  (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_Commit)
  (COMMAND_ATTRIBUTES)(CC_Commit * // 0x018B
  (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicyPassword)
  (COMMAND_ATTRIBUTES)(CC_PolicyPassword * // 0x018C
  (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_ZGen_2Phase)
  (COMMAND_ATTRIBUTES)(CC_ZGen_2Phase * // 0x018D
  (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_EC_Ephemeral)
  (COMMAND_ATTRIBUTES)(CC_EC_Ephemeral * // 0x018E
  (IS_IMPLEMENTED+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_PolicyNvWritten)
  (COMMAND_ATTRIBUTES)(CC_PolicyNvWritten * // 0x018F
  (IS_IMPLEMENTED+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_PolicyTemplate)
  (COMMAND_ATTRIBUTES)(CC_PolicyTemplate * // 0x0190
  (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_CreateLoaded)
  (COMMAND_ATTRIBUTES)(CC_CreateLoaded * // 0x0191
  (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+PP_COMMAND+ENCRYPT_2+R_HANDLE)),
#endif
#if (PAD_LIST || CC_PolicyAuthorizeNV)
  (COMMAND_ATTRIBUTES)(CC_PolicyAuthorizeNV * // 0x0192
  (IS_IMPLEMENTED+HANDLE_1_USER+ALLOW_TRIAL)),
#endif
#if (PAD_LIST || CC_EncryptDecrypt2)
  (COMMAND_ATTRIBUTES)(CC_EncryptDecrypt2 * // 0x0193
  (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_USER+ENCRYPT_2)),
#endif
#if (PAD_LIST || CC_AC_GetCapability)
  (COMMAND_ATTRIBUTES)(CC_AC_GetCapability * // 0x0194
  (IS_IMPLEMENTED)),
#endif
#if (PAD_LIST || CC_AC_Send)
  (COMMAND_ATTRIBUTES)(CC_AC_Send * // 0x0195
  (IS_IMPLEMENTED+DECRYPT_2+HANDLE_1_DUP+HANDLE_2_USER)),
#endif
#ifdef

#if (PAD_LIST || CC_Policy_AC_SendSelect)
    (COMMAND_ATTRIBUTES)(CC_Policy_AC_SendSelect * // 0x0196
    (IS_IMPLEMENTED+DECRYPT_2+ALLOW_TRIAL)),
#endif

#if (PAD_LIST || CC_Vendor_TCG_Test)
    (COMMAND_ATTRIBUTES)(CC_Vendor_TCG_Test * // 0x0000
    (IS_IMPLEMENTED+DECRYPT_2+ENCRYPT_2)),
#endif

0

#endif // _COMMAND_CODE_ATTRIBUTES_
5.5 CommandAttributes.h

The attributes defined in this file are produced by the parser that creates the structure definitions from Part 3. The attributes are defined in that parser and should track the attributes being tested in CommandCodeAttributes.c. Generally, when an attribute is added to this list, new code will be needed in CommandCodeAttributes.c to test it.

1 #ifndef COMMAND_ATTRIBUTES_H
2 #define COMMAND_ATTRIBUTES_H
3 typedef UINT16 COMMAND_ATTRIBUTES;
4 #define NOT_IMPLEMENTED (COMMAND_ATTRIBUTES)(0)
5 #define ENCRYPT_2 ((COMMAND_ATTRIBUTES)1 << 0)
6 #define ENCRYPT_4 ((COMMAND_ATTRIBUTES)1 << 1)
7 #define DECRYPT_2 ((COMMAND_ATTRIBUTES)1 << 2)
8 #define DECRYPT_4 ((COMMAND_ATTRIBUTES)1 << 3)
9 #define HANDLE_1_USER ((COMMAND_ATTRIBUTES)1 << 4)
10 #define HANDLE_1_ADMIN ((COMMAND_ATTRIBUTES)1 << 5)
11 #define HANDLE_1_DUP ((COMMAND_ATTRIBUTES)1 << 6)
12 #define HANDLE_2_USER ((COMMAND_ATTRIBUTES)1 << 7)
13 #define PP_COMMAND ((COMMAND_ATTRIBUTES)1 << 8)
14 #define IS_IMPLEMENTED ((COMMAND_ATTRIBUTES)1 << 9)
15 #define NO_SESSIONS ((COMMAND_ATTRIBUTES)1 << 10)
16 #define NV_COMMAND ((COMMAND_ATTRIBUTES)1 << 11)
17 #define PP_REQUIRED ((COMMAND_ATTRIBUTES)1 << 12)
18 #define R_HANDLE ((COMMAND_ATTRIBUTES)1 << 13)
19 #define ALLOW_TRIAL ((COMMAND_ATTRIBUTES)1 << 14)
20 #endif // COMMAND_ATTRIBUTES_H
5.6 CommandDispatchData.h

This file should only be included by CommandCodeAttributes.c

```c
#ifdef _COMMAND_TABLE_DISPATCH_

Define the stop value

#define END_OF_LIST 0xff
#define ADD_FLAG 0x80
```

The UnmarshalArray() contains the dispatch functions for the unmarshaling code. The defines in this array are used to make it easier to cross reference the unmarshaling values in the types array of each command.

```c
const UNMARSHAL_t UnmarshalArray[] = {
#define TPMI_DH_CONTEXT_H_UNMARSHAL (UNMARSHAL_t)TPMI_DH_CONTEXT_Unmarshal,
#define TPMI_RH_AC_H_UNMARSHAL (TPMI_DH_CONTEXT_H_UNMARSHAL + 1)
#define TPMI_RH_CLEAR_H_UNMARSHAL (TPMI_RH_AC_H_UNMARSHAL + 1)
#define TPMI_RH_HIERARCHY_AUTH_H_UNMARSHAL (TPMI_RH_CLEAR_H_UNMARSHAL + 1)
#define TPMI_RH_LOCKOUT_H_UNMARSHAL (TPMI_RH_HIERARCHY_AUTH_H_UNMARSHAL + 1)
#define TPMI_RH_NV_AUTH_H_UNMARSHAL (TPMI_RH_LOCKOUT_H_UNMARSHAL + 1)
#define TPMI_RH_NV_INDEX_H_UNMARSHAL (TPMI_RH_NV_AUTH_H_UNMARSHAL + 1)
#define TPMI_RH_PLATFORM_H_UNMARSHAL (TPMI_RH_NV_INDEX_H_UNMARSHAL + 1)
#define TPMI_RH_PROVISION_H_UNMARSHAL (TPMI_RH_PLATFORM_H_UNMARSHAL + 1)
#define TPMI_SH_HMAC_H_UNMARSHAL (TPMI_SH_POLICY_H_UNMARSHAL + 1)
#define TPMI_SH_POLICY_H_UNMARSHAL (TPMI_SH_HMAC_H_UNMARSHAL + 1)
#define TPMI_DH_ENTITY_H_UNMARSHAL (TPMI_SH_POLICY_H_UNMARSHAL + 1)
#define TPMI_DH_OBJECT_H_UNMARSHAL (TPMI_DH_ENTITY_H_UNMARSHAL + 1)
#define TPMI_DH_PARENT_H_UNMARSHAL (TPMI_DH_OBJECT_H_UNMARSHAL + 1)
#define TPMI_DH_PCR_H_UNMARSHAL (TPMI_DH_PARENT_H_UNMARSHAL + 1)
#define TPMI_RH_ENDORSEMENT_H_UNMARSHAL (TPMI_DH_PCR_H_UNMARSHAL + 1)
#define TPMI_RH_HIERARCHY_H_UNMARSHAL (TPMI_RH_ENDORSEMENT_H_UNMARSHAL + 1)

// HANDLE_FIRST_FLAG_TYPE is the first handle that needs a flag when called.
#define HANDLE_FIRST_FLAG_TYPE (TPMI_RH_HIERARCHY_H_UNMARSHAL + 1)
#define PARAMETER_FIRST_TYPE (TPMI_RH_HIERARCHY_H_UNMARSHAL + 1)
#define TPM2B_DATA_P_UNMARSHAL (TPM2B_DATA_P_UNMARSHAL + 1)
#define TPM2B_DIGEST_P_UNMARSHAL (TPM2B_DIGEST_P_UNMARSHAL + 1)
#define TPM2B_ECC_PARAMETER_P_UNMARSHAL (TPM2B_ECC_PARAMETER_P_UNMARSHAL + 1)
#define TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL (TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL + 1)

// PARAMETER_FIRST_TYPE marks the end of the handle list.
#define PARAMETER_FIRST_TYPE (TPMI_RH_HIERARCHY_H_UNMARSHAL + 1)
#define TPM2B_DATA_P_UNMARSHAL (TPM2B_DATA_P_UNMARSHAL + 1)
#define TPM2B_DIGEST_P_UNMARSHAL (TPM2B_DIGEST_P_UNMARSHAL + 1)
#define TPM2B_ECC_PARAMETER_P_UNMARSHAL (TPM2B_ECC_PARAMETER_P_UNMARSHAL + 1)
#define TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL (TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL + 1)
```

```c}
#endif
```

```c
#endif
```
(UNMARSHAL_t)TPM2B_ENCRYPTED_SECRET_Unmarshal,
#define TPM2B_EVENT_P_UNMARSHAL (TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_EVENT_Unmarshal,
#define TPM2B_ID_OBJECT_P_UNMARSHAL (TPM2B_EVENT_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_ID_OBJECT_Unmarshal,
#define TPM2B_IV_P_UNMARSHAL (TPM2B_ID_OBJECT_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_IV_Unmarshal,
#define TPM2B_MAX_BUFFER_P_UNMARSHAL (TPM2B_IV_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_MAX_BUFFER_Unmarshal,
#define TPM2B_MAX_NV_BUFFER_P_UNMARSHAL (TPM2B_MAX_BUFFER_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_MAX_NV_BUFFER_Unmarshal,
#define TPM2B_NAME_P_UNMARSHAL (TPM2B_MAX_NV_BUFFER_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_NAME_Unmarshal,
#define TPM2B_NV_PUBLIC_P_UNMARSHAL (TPM2B_NAME_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_NV_PUBLIC_Unmarshal,
#define TPM2B_PRIVATE_P_UNMARSHAL (TPM2B_NV_PUBLIC_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_PRIVATE_Unmarshal,
#define TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL (TPM2B_PRIVATE_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_PUBLIC_KEY_RSA_Unmarshal,
#define TPM2B_SENSITIVE_P_UNMARSHAL (TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_SENSITIVE_Unmarshal,
#define TPM2B_SENSITIVE_CREATE_P_UNMARSHAL (TPM2B_SENSITIVE_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_SENSITIVE_CREATE_Unmarshal,
#define TPM2B_SENSITIVE_DATA_P_UNMARSHAL (TPM2B_SENSITIVE_CREATE_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_SENSITIVE_DATA_Unmarshal,
#define TPM2B_TEMPLATE_P_UNMARSHAL (TPM2B_SENSITIVE_DATA_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_TEMPLATE_Unmarshal,
#define TPM2B_TIMEOUT_P_UNMARSHAL (TPM2B_TEMPLATE_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM2B_TIMEOUT_Unmarshal,
#define TPMI_DH_CONTEXT_P_UNMARSHAL (TPM2B_TIMEOUT_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMI_DH_CONTEXT_Unmarshal,
#define TPMI_DH_PERSISTENT_P_UNMARSHAL (TPMI_DH_CONTEXT_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMI_DH_PERSISTENT_Unmarshal,
#define TPMI_ECC_CURVE_P_UNMARSHAL (TPMI_DH_PERSISTENT_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMI_ECC_CURVE_Unmarshal,
#define TPMI_YES_NO_P_UNMARSHAL (TPMI_ECC_CURVE_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMI_YES_NO_Unmarshal,
#define TPML_ALG_P_UNMARSHAL (TPMI_YES_NO_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPML_ALG_Unmarshal,
#define TPML_CC_P_UNMARSHAL (TPML_ALG_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPML_CC_Unmarshal,
#define TPML_DIGEST_P_UNMARSHAL (TPML_CC_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPML_DIGEST_Unmarshal,
#define TPML_DIGEST_VALUES_P_UNMARSHAL (TPML_DIGEST_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPML_DIGEST_VALUES_Unmarshal,
#define TPML_PCR_SELECTION_P_UNMARSHAL (TPML_DIGEST_VALUES_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPML_PCR_SELECTION_Unmarshal,
#define TPMS_CONTEXT_P_UNMARSHAL (TPML_PCR_SELECTION_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMS_CONTEXT_Unmarshal,
#define TPMT_PUBLIC_PARMS_P_UNMARSHAL (TPMS_CONTEXT_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMT_PUBLIC_PARMS_Unmarshal,
#define TPMT_TK_AUTH_P_UNMARSHAL (TPMT_PUBLIC_PARMS_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMT_TK_AUTH_Unmarshal,
#define TPMT_TK_CREATION_P_UNMARSHAL (TPMT_TK_AUTH_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMT_TK_CREATION_Unmarshal,
#define TPMT_TK_HASHCHECK_P_UNMARSHAL (TPMT_TK_CREATION_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMT_TK_HASHCHECK_Unmarshal,
#define TPMT_TK_VERIFIED_P_UNMARSHAL (TPMT_TK_HASHCHECK_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPMT_TK_VERIFIED_Unmarshal,
#define TPM_AT_P_UNMARSHAL (TPMT_TK_VERIFIED_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM_AT_Unmarshal,
#define TPM_CAP_P_UNMARSHAL (TPM_AT_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM_CAP_Unmarshal,
#define TPM_CLOCK_ADJUST_P_UNMARSHAL (TPM_CAP_P_UNMARSHAL + 1)
(UNMARSHAL_t)TPM_CLOCK_ADJUST_Unmarshal,
#define TPM_EOL_P_UNMARSHAL (TPM_CLOCK_ADJUST_P_UNMARSHAL + 1)
#define TPM_EO_Unmarshal     (UNMARSHAL_t)TPM_EO_Unmarshal,
#define TPM_SE_P_UNMARSHAL   (UNMARSHAL_t)TPM_SE_Unmarshal,
#define TPM_SU_P_UNMARSHAL   (UNMARSHAL_t)TPM_SU_Unmarshal,
#define UINT16_P_UNMARSHAL   (UNMARSHAL_t)UINT16_Unmarshal,
#define UINT32_P_UNMARSHAL   (UNMARSHAL_t)UINT32_Unmarshal,
#define UINT64_P_UNMARSHAL   (UNMARSHAL_t)UINT64_Unmarshal,
#define UINT8_P_UNMARSHAL    (UNMARSHAL_t)UINT8_Unmarshal,
#define PARAMETER_FIRST_FLAG_TYPE (UNMARSHAL_t)PARAMETER_FIRST_FLAG_TYPE
#define TPM2B_PUBLIC_P_UNMARSHAL (UNMARSHAL_t)TPM2B_PUBLIC_Unmarshal,
#define TPM2B_CREATION_DATA_P_MARSHAL (UNMARSHAL_t)TPM2B_CREATION_DATA_Marshal,
#define TPM2B_DATA_P_MARSHAL (UNMARSHAL_t)TPM2B_DATA_Marshal,
#define TPM2B_DIGEST_P_MARSHAL (UNMARSHAL_t)TPM2B_DIGEST_Marshal,
#define TPM2B_ECC_POINT_P_MARSHAL (UNMARSHAL_t)TPM2B_ECC_POINT_Marshal,
#define TPM2B_ENCRYPTED_SECRET_P_MARSHAL (UNMARSHAL_t)TPM2B_ENCRYPTED_SECRET_Marshal,

const MARSHAL_t MarshalArray[] = {
#define UINT32_H_MARSHAL 0
#define TPM2B_PUBLIC_P_MARSHAL (MARSHAL_t)TPM2B_PUBLIC_Unmarshal,
#define TPM2B_CREATION_DATA_P_MARSHAL (MARSHAL_t)TPM2B_CREATION_DATA_Marshal,
#define TPM2B_DATA_P_MARSHAL (MARSHAL_t)TPM2B_DATA_Marshal,
#define TPM2B_DIGEST_P_MARSHAL (MARSHAL_t)TPM2B_DIGEST_Marshal,
#define TPM2B_ECC_POINT_P_MARSHAL (MARSHAL_t)TPM2B_ECC_POINT_Marshal,
#define TPM2B_ENCRYPTED_SECRET_P_MARSHAL (MARSHAL_t)TPM2B_ENCRYPTED_SECRET_Marshal,

The MarshalArray() contains the dispatch functions for the marshaling code. The defines in this array are used to make it easier to cross reference the marshaling values in the types array of each command.
```c
#define TPM2B_ID_OBJECT_P_MARSHAL ((TPM2B_ENCRYPTED_SECRET_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_ID_OBJECT_Marshal,
#define TPM2B_IV_P_MARSHAL ((TPM2B_ID_OBJECT_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_IV_Marshal,
#define TPM2B_MAX_BUFFER_P_MARSHAL ((TPM2B_MAX_BUFFER_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_MAX_BUFFER_Marshal,
#define TPM2B_MAX_NV_BUFFER_P_MARSHAL ((TPM2B_MAX_NV_BUFFER_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_MAX_NV_BUFFER_Marshal,
#define TPM2B_NAME_P_MARSHAL ((TPM2B_MAX_NV_BUFFER_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_NAME_Marshal,
#define TPM2B_NV_PUBLIC_P_MARSHAL ((TPM2B_NAME_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_NV_PUBLIC_Marshal,
#define TPM2B_PRIVATE_P_MARSHAL ((TPM2B_NV_PUBLIC_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_PRIVATE_Marshal,
#define TPM2B_PUBLIC_P_MARSHAL ((TPM2B_PRIVATE_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_PUBLIC_Marshal,
#define TPM2B_PUBLIC_KEY_RSA_P_MARSHAL ((TPM2B_PUBLIC_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_PUBLIC_KEY_RSA_Marshal,
#define TPM2B_SENSITIVE_DATA_P_MARSHAL ((TPM2B_PUBLIC_KEY_RSA_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_SENSITIVE_DATA_Marshal,
#define TPM2B_TIMEOUT_P_MARSHAL ((TPM2B_SENSITIVE_DATA_P_MARSHAL + 1)
(MARSHAL_t)TPM2B_TIMEOUT_Marshal,
#define UINT8_P_MARSHAL ((TPM2B_TIMEOUT_P_MARSHAL + 1)
(MARSHAL_t)UINT8_Marshal,
#define TPML_AC_CAPABILITIES_P_MARSHAL ((UINT8_P_MARSHAL + 1)
(MARSHAL_t)TPML_AC_CAPABILITIES_Marshal,
#define TPML_ALG_P_MARSHAL ((TPML_AC_CAPABILITIES_P_MARSHAL + 1)
(MARSHAL_t)TPML_ALG_Marshal,
#define TPML_DIGEST_P_MARSHAL ((TPML_ALG_P_MARSHAL + 1)
(MARSHAL_t)TPML_DIGEST_Marshal,
#define TPML_DIGEST_VALUES_P_MARSHAL ((TPML_DIGEST_P_MARSHAL + 1)
(MARSHAL_t)TPML_DIGEST_VALUES_Marshal,
#define TPM_AC_OUTPUT_P_MARSHAL ((TPML_DIGEST_VALUES_P_MARSHAL + 1)
(MARSHAL_t)TPMS_AC_OUTPUT_Marshal,
#define TPMS_ALGORITHM_DETAIL_ECC_P_MARSHAL ((TPMS_AC_OUTPUT_P_MARSHAL + 1)
(MARSHAL_t)TPMS_ALGORITHM_DETAIL_ECC_Marshal,
#define TPMS_CAPABILITY_DATA_P_MARSHAL ((TPMS_ALGORITHM_DETAIL_ECC_P_MARSHAL + 1)
(MARSHAL_t)TPMS_CAPABILITY_DATA_Marshal,
#define TPMS_CONTEXT_P_MARSHAL ((TPMS_CAPABILITY_DATA_P_MARSHAL + 1)
(MARSHAL_t)TPMS_CONTEXT_Marshal,
#define TPMS_TIME_INFO_P_MARSHAL ((TPMS_CONTEXT_P_MARSHAL + 1)
(MARSHAL_t)TPMS_TIME_INFO_Marshal,
#define TPMT_HA_P_MARSHAL ((TPMS_TIME_INFO_P_MARSHAL + 1)
(MARSHAL_t)TPMT_HA_Marshal,
#define TPMT_SIGNATURE_P_MARSHAL ((TPMT_HA_P_MARSHAL + 1)
(MARSHAL_t)TPMT_SIGNATURE_Marshal,
#define TPMT_TK_AUTH_P_MARSHAL ((TPMT_SIGNATURE_P_MARSHAL + 1)
(MARSHAL_t)TPMT_TK_AUTH_Marshal,
#define TPMT_TK_CREATION_P_MARSHAL ((TPMT_TK_AUTH_P_MARSHAL + 1)
(MARSHAL_t)TPMT_TK_CREATION_Marshal,
#define TPMT_TK_HASHCHECK_P_MARSHAL ((TPMT_TK_CREATION_P_MARSHAL + 1)
(MARSHAL_t)TPMT_TK_HASHCHECK_Marshal,
#define TPMT_TK_VERIFIED_P_MARSHAL ((TPMT_TK_HASHCHECK_P_MARSHAL + 1)
(MARSHAL_t)TPMT_TK_VERIFIED_Marshal,
#define UINT32_P_MARSHAL ((TPMT_TK_VERIFIED_P_MARSHAL + 1)
(MARSHAL_t)UINT32_Marshal,
#define UINT16_P_MARSHAL ((UINT32_P_MARSHAL + 1)
(MARSHAL_t)UINT16_Marshal,

// RESPONSE_PARAMETER_LAST_TYPE is the end of the response parameter list.
#define RESPONSE_PARAMETER_LAST_TYPE (UINT16_P_MARSHAL)
```
This list of aliases allows the types in the _COMMAND_DESCRIPTOR_T to match the types in the command/response templates of part 3.

```c
#define INT32_P_UNMARSHAL UINT32_P_UNMARSHAL
#define TPM2B_AUTH_P_UNMARSHAL TPM2B_DIGEST_P_UNMARSHAL
#define TPM2B_NONCE_P_UNMARSHAL TPM2B_DIGEST_P_UNMARSHAL
#define TPM2B_OPERAND_P_UNMARSHAL TPM2B_DIGEST_P_UNMARSHAL
#define TPM_LOCALITY_P_UNMARSHAL UINT8_P_UNMARSHAL
#define TPM_CC_P_UNMARSHAL UINT32_P_UNMARSHAL
#define TPM_ID_CONTEXT_H_MARSHAL UINT32_H_MARSHAL
#define TPM_ID_OBJECT_H_MARSHAL UINT32_H_MARSHAL
#define TPM_ID_AUTH_SESSION_H_MARSHAL UINT32_H_MARSHAL
#define TPM_HANDLE_H_MARSHAL UINT32_H_MARSHAL
#define TPM2B_NONCE_P_MARSHAL TPM2B_DIGEST_P_MARSHAL
#define TPMI_YES_NO_P_MARSHAL UINT8_P_MARSHAL
#define TPM_RC_P_MARSHAL UINT32_P_MARSHAL
#if CC_Startup
#include "Startup_fp.h"
typedef TPM_RC (Startup_Entry)
    *(Startup_In)
#endif
typedef const struct
    {
        Startup_Entry           *entry;
        UINT16                  inSize;
        UINT16                  outSize;
        UINT16                  offsetOfTypes;
        BYTE                    types[3];
    } Startup_COMMAND_DESCRIPTOR_t;
Startup_COMMAND_DESCRIPTOR_t _StartupData = {
    /* entry         */     &TPM2_Startup,
    /* inSize        */     (UINT16)(sizeof(Startup_In)),
    /* outSize       */     0,
    /* offsetOfTypes */     offsetOf(Startup_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */     // No parameter offsets;
    /* types         */     {TPM_SU_P_UNMARSHAL, END_OF_LIST, END_OF_LIST}
};
#define _StartupDataAddress (&_StartupData)
#else
#define _StartupDataAddress 0
#endif // CC_Startup
#if CC_Shutdown
#include "Shutdown_fp.h"
typedef TPM_RC (Shutdown_Entry)
    *(Shutdown_In)
#endif
typedef const struct
    {
        Shutdown_Entry          *entry;
        UINT16                  inSize;
        UINT16                  outSize;
        UINT16                  offsetOfTypes;
        BYTE                    types[3];
    } Shutdown_COMMAND_DESCRIPTOR_t;
Shutdown_COMMAND_DESCRIPTOR_t _ShutdownData = {
    /* entry         */     &TPM2_Shutdown,
    /* inSize        */     (UINT16)(sizeof(Shutdown_In)),
    /* outSize       */     0,
    /* offsetOfTypes */     offsetOf(Shutdown_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */     // No parameter offsets;
    /* types         */     {TPM_SU_P_UNMARSHAL, END_OF_LIST, END_OF_LIST}
};
#define _ShutdownDataAddress (&_ShutdownData)
```
#else
#define _ShutdownDataAddress 0
#endif

if CC_SelfTest
#include "SelfTest_fp.h"
typedef TPM_RC (SelfTest_Entry)(
    SelfTest_In *in
);
typedef const struct {
    SelfTest_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    BYTE types[3];
} SelfTest_COMMAND_DESCRIPTOR_t;
SelfTest_COMMAND_DESCRIPTOR_t _SelfTestData = {
    /* entry */ &TPM2_SelfTest,
    /* inSize */ (UINT16)(sizeof(SelfTest_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(SelfTest_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ // No parameter offsets;
    /* types */ {TPMI_YES_NO_P_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};
#define _SelfTestDataAddress (&_SelfTestData)
#else
#define _SelfTestDataAddress 0
#endif

if CC_IncrementalSelfTest
#include "IncrementalSelfTest_fp.h"
typedef TPM_RC (IncrementalSelfTest_Entry)(
    IncrementalSelfTest_In *in,
    IncrementalSelfTest_Out *out
);
typedef const struct {
    IncrementalSelfTest_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    BYTE types[4];
} IncrementalSelfTest_COMMAND_DESCRIPTOR_t;
IncrementalSelfTest_COMMAND_DESCRIPTOR_t _IncrementalSelfTestData = {
    /* entry */ &TPM2_IncrementalSelfTest,
    /* inSize */ (UINT16)(sizeof(IncrementalSelfTest_In)),
    /* outSize */ (UINT16)(sizeof(IncrementalSelfTest_Out)),
    /* offsetOfTypes */ offsetof(IncrementalSelfTest_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ // No parameter offsets;
    /* types */ {TPML_ALG_P_UNMARSHAL,
        END_OF_LIST,
        TPML_ALG_P_MARSHAL,
        END_OF_LIST}
};
#define _IncrementalSelfTestDataAddress (&_IncrementalSelfTestData)
#else
#define _IncrementalSelfTestDataAddress 0
#endif

if CC_GetTestResult
#include "GetTestResult_fp.h"
typedef TPM_RC (GetTestResult_Entry)(
    GetTestResult_Out *out
);
typedef const struct {
    GetTestResult_Entry *entry;
    UINT16 inSize;
} GetTestResult_COMMAND_DESCRIPTOR_t;
GetTestResult_COMMAND_DESCRIPTOR_t _GetTestResultData = {
};
#define _GetTestResultDataAddress (&_GetTestResultData)
#else
#define _GetTestResultDataAddress 0
#endif

UINT16 outSize;
UINT16 offsetOfTypes;
UINT16 paramOffsets[1];
BYTE types[4];

GetTestResult_COMMAND_DESCRIPTOR_t GetTestResultData = {
/* entry */ &TPM2_GetTestResult,
/* inSize */ 0,
/* outSize */ (UINT16)(sizeof(GetTestResult_Out)),
/* offsetOfTypes */ offsetof(GetTestResult_COMMAND_DESCRIPTOR_t, types),
/* offsets */ {(UINT16)(offsetof(GetTestResult_Out, testResult))},
/* types */ {END_OF_LIST,
TPM2B_MAX_BUFFER_P_MARSHAL,
TPM_RC_P_MARSHAL,
END_OF_LIST}
};

#define _GetTestResultDataAddress (&_GetTestResultData)
#else
#define _GetTestResultDataAddress 0
#endif
// CC_GetTestResult

#if CC_StartAuthSession
#include "StartAuthSession_fp.h"
typedef TPM_RC (StartAuthSession_Entry)(
StartAuthSession_In *in,
StartAuthSession_Out *out
);
typedef const struct {
StartAuthSession_Entry *entry;
UINT16 inSize;
UINT16 outSize;
UINT16 offsetOfTypes;
UINT16 paramOffsets[7];
BYTE types[11];
} StartAuthSession_COMMAND_DESCRIPTOR_t;
StartAuthSession_COMMAND_DESCRIPTOR_t _StartAuthSessionData = {
/* entry */ &TPM2_StartAuthSession,
/* inSize */ (UINT16)(sizeof(StartAuthSession_In)),
/* outSize */ (UINT16)(sizeof(StartAuthSession_Out)),
/* offsetOfTypes */ offsetof(StartAuthSession_COMMAND_DESCRIPTOR_t, types),
/* offsets */ {(UINT16)(offsetof(StartAuthSession_In, bind)),
(UINT16)(offsetof(StartAuthSession_In, nonceCaller)),
(UINT16)(offsetof(StartAuthSession_In, encryptedSalt)),
(UINT16)(offsetof(StartAuthSession_In, sessionType)),
(UINT16)(offsetof(StartAuthSession_In, symmetric)),
(UINT16)(offsetof(StartAuthSession_In, authHash)),
(UINT16)(offsetof(StartAuthSession_Out, nonceTPM))},
/* types */ {TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
TPMI_DH_ENTITY_H_UNMARSHAL + ADD_FLAG,
TPM2B_NONCE_P_UNMARSHAL,
TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL,
TPM_SE_P_UNMARSHAL,
TPMT_SYM_DEF_P_UNMARSHAL + ADD_FLAG,
TPMT_ALG_HASH_P_UNMARSHAL,
END_OF_LIST,
TPMI_SH_AUTH_SESSION_H_MARSHAL,
TPM2B_NONCE_P_MARSHAL,
END_OF_LIST}
};
#define _StartAuthSessionDataAddress (&_StartAuthSessionData)
#else
#define _StartAuthSessionDataAddress 0
#endif
// CC_StartAuthSession

#if CC_PolicyRestart
#include "PolicyRestart_fp.h"
typedef TPM_RC (PolicyRestart_Entry)(
PolicyRestart_In *in
);
typedef const struct {
    PolicyRestart_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    BYTE types[3];
} PolicyRestart_COMMAND_DESCRIPTOR_t;

PolicyRestart_COMMAND_DESCRIPTOR_t _PolicyRestartData = {
    /* entry */ &TPM2_PolicyRestart,
    /* inSize */ (UINT16)(sizeof(PolicyRestart_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetOf(PolicyRestart_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ // No parameter offsets:
    /* types */ {TPMI_SH_POLICY_H_UNMARSHAL,
    END_OF_LIST,
    END_OF_LIST}
};

#define _PolicyRestartDataAddress (&_PolicyRestartData)
#endif // CC_PolicyRestart

#ifdef CC_Create
#include "Create_fp.h"

typedef TPM_RC (Create_Entry)(
    Create_In *in,
    Create_Out *out,
);}

typedef const struct {
    Create_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[8];
    BYTE types[12];
} Create_COMMAND_DESCRIPTOR_t;

Create_COMMAND_DESCRIPTOR_t _CreateData = {
    /* entry */ &TPM2_Create,
    /* inSize */ (UINT16)(sizeof(Create_In)),
    /* outSize */ (UINT16)(sizeof(Create_Out)),
    /* offsetOfTypes */ (UINT16)(offsetof(Create_COMMAND_DESCRIPTOR_t, types)),
    /* offsets */ {(UINT16)(offsetof(Create_In, inSensitive)),
    (UINT16)(offsetof(Create_In, inPublic)),
    (UINT16)(offsetof(Create_In, outsideInfo)),
    (UINT16)(offsetof(Create_In, creationPCR)),
    (UINT16)(offsetof(Create_Out, outPublic)),
    (UINT16)(offsetof(Create_Out, creationData)),
    (UINT16)(offsetof(Create_Out, creationHash)),
    (UINT16)(offsetof(Create_Out, creationTicket))},
    /* types */ {TPMI_DH_OBJECT_H_UNMARSHAL,
    TPM2B_SENSITIVE_CREATE_P_UNMARSHAL,
    TPM2B_PUBLIC_P_UNMARSHAL,
    TPM2B_DATA_P_UNMARSHAL,
    TPML_PCR_SELECTION_P_UNMARSHAL,
    END_OF_LIST,
    TPM2B_PRIVATE_P_MARSHAL,
    TPM2B_PUBLIC_P_MARSHAL,
    TPM2B_CREATION_DATA_P_MARSHAL,
    TPM2B_DIGEST_P_MARSHAL,
    TPMT_TK_CREATION_P_MARSHAL,
    END_OF_LIST}
};

#define _CreateDataAddress (&_CreateData)
#else
#define _CreateDataAddress 0
#endif // CC_Create
#if CC_Load
#include "Load_fp.h"
typedef TPM_RC (Load_Entry)(
    Load_In                     *in,
    Load_Out                    *out
);
typedef const struct {
    Load_Entry              *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[3];
    BYTE                    types[7];
} Load_COMMAND_DESCRIPTOR_t;
Load_COMMAND_DESCRIPTOR_t _LoadData = {
    /* entry         */ &TPM2_Load,
    /* inSize        */ (UINT16)(sizeof(Load_In)),
    /* outSize       */ (UINT16)(sizeof(Load_Out)),
    /* offsetOfTypes */ (UINT16)(offsetof(Load_COMMAND_DESCRIPTOR_t, types)),
    /* offsets       */ {
        (UINT16)(offsetof(Load_In, inPrivate)),
        (UINT16)(offsetof(Load_In, inPublic)),
        (UINT16)(offsetof(Load_Out, name))},
    /* types         */ {TPMI_DH_OBJECT_H_UNMARSHAL,
        TPM2B_PRIVATE_P_UNMARSHAL,
        TPM2B_PUBLIC_P_UNMARSHAL,
        END_OF_LIST,
        TPM_HANDLE_H_MARSHAL,
        TPM2B_NAME_P_MARSHAL,
        END_OF_LIST}
};
#define _LoadDataAddress (&_LoadData)
#else
#define _LoadDataAddress 0
#endif // CC_Load
#endif // CC_LoadExternal
#include "LoadExternal_fp.h"
typedef TPM_RC (LoadExternal_Entry)(
    LoadExternal_In             *in,
    LoadExternal_Out            *out
);
typedef const struct {
    LoadExternal_Entry      *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[3];
    BYTE                    types[7];
} LoadExternal_COMMAND_DESCRIPTOR_t;
LoadExternal_COMMAND_DESCRIPTOR_t _LoadExternalData = {
    /* entry         */ &TPM2_LoadExternal,
    /* inSize        */ (UINT16)(sizeof(LoadExternal_In)),
    /* outSize       */ (UINT16)(sizeof(LoadExternal_Out)),
    /* offsetOfTypes */ (UINT16)(offsetof(LoadExternal_COMMAND_DESCRIPTOR_t, types)),
    /* offsets       */ {
        (UINT16)(offsetof(LoadExternal_In, inPublic)),
        (UINT16)(offsetof(LoadExternal_In, hierarchy)),
        (UINT16)(offsetof(LoadExternal_Out, name))},
    /* types         */ {TPM2B_SENSITIVE_P_UNMARSHAL,
        TPM2B_PUBLIC_P_UNMARSHAL + ADD_FLAG,
        TPM1_RH_HIERARCHY_P_UNMARSHAL + ADD_FLAG,
        END_OF_LIST,
        TPM_HANDLE_H_MARSHAL,
        TPM2B_NAME_P_MARSHAL,
        END_OF_LIST}
};
#define _LoadExternalDataAddress (&_LoadExternalData)
ge else
typedef TPM_RC (ReadPublic_Entry)(
    ReadPublic_In *in,
    ReadPublic_Out *out
);

typedef const struct {
    ReadPublic_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[2];
    BYTE types[6];
} ReadPublic_COMMAND_DESCRIPTOR_t;

ReadPublic_COMMAND_DESCRIPTOR_t _ReadPublicData = {
    /* entry */ &TPM2_ReadPublic,
    /* inSize */ (UINT16)(sizeof(ReadPublic_In)),
    /* outSize */ (UINT16)(sizeof(ReadPublic_Out)),
    /* offsetOfTypes */ offsetof(ReadPublic_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {{(UINT16)(offsetof(ReadPublic_Out, name))},
        (UINT16)(offsetof(ReadPublic_Out, qualifiedName))},
    /* types */ {TPMI_DH_OBJECT_H_UNMARSHAL,
        END_OF_LIST,
        TPM2B_PUBLIC_P_MARSHAL,
        TPM2B_NAME_P_MARSHAL,
        TPM2B_NAME_P_MARSHAL,
        END_OF_LIST}
};

#define _ReadPublicDataAddress (&_ReadPublicData)

typedef TPM_RC (ActivateCredential_Entry)(
    ActivateCredential_In *in,
    ActivateCredential_Out *out
);

typedef const struct {
    ActivateCredential_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[3];
    BYTE types[7];
} ActivateCredential_COMMAND_DESCRIPTOR_t;

ActivateCredential_COMMAND_DESCRIPTOR_t _ActivateCredentialData = {
    /* entry */ &TPM2_ActivateCredential,
    /* inSize */ (UINT16)(sizeof(ActivateCredential_In)),
    /* outSize */ (UINT16)(sizeof(ActivateCredential_Out)),
    /* offsetOfTypes */ offsetof(ActivateCredential_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {{(UINT16)(offsetof(ActivateCredential_In, keyHandle))},
        (UINT16)(offsetof(ActivateCredential_In, credentialBlob))},
    /* types */ {TPMI_DH_OBJECT_H_UNMARSHAL,
        TPM2B_PUBLIC_P_MARSHAL,
        TPM2B_ID_OBJECT_P_UNMARSHAL,
        TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL,
        TPM2B_DIGEST_P_MARSHAL,
        END_OF_LIST}
};
#define _ActivateCredentialDataAddress (&_ActivateCredentialData)
#else
#define _ActivateCredentialDataAddress 0
#endif

// CC_ActivateCredential

#define _MakeCredentialDataAddress (&_MakeCredentialData)
#else
#define _MakeCredentialDataAddress 0
#endif

// CC_MakeCredential

#define _UnsealDataAddress (&_UnsealData)
#else
#define _UnsealDataAddress 0
#endif

// CC_Unseal

typedef TPM_RC (MakeCredential_Entry)(
    MakeCredential_In  *in,
    MakeCredential_Out *out
);
typedef const struct {
    MakeCredential_Entry    *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[3];
    BYTE                    types[7];
} MakeCredential_COMMAND_DESCRIPTOR_t;

MakeCredential_COMMAND_DESCRIPTOR_t _MakeCredentialData = {
    /* entry         */ &TPM2_MakeCredential,
    /* inSize        */ (UINT16)(sizeof(MakeCredential_In)),
    /* outSize       */ (UINT16)(sizeof(MakeCredential_Out)),
    /* offsetOfTypes */ offsetOf(MakeCredential_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {
        (UINT16)(offsetof(MakeCredential_In, credential)),
        (UINT16)(offsetof(MakeCredential_In, objectName)),
        (UINT16)(offsetof(MakeCredential_Out, secret))},
    /* types         */ {
        TPM2B_DIGEST_P_UNMARSHAL,
        TPM2B_NAME_P_UNMARSHAL,
        TPM2B_NAME_P_MARSHAL,
        TPM2B_ID_OBJECT_P_MARSHAL,
    END_OF_LIST}
};

#define _MakeCredentialDataAddress (&_MakeCredentialData)
#else
#define _MakeCredentialDataAddress 0
#endif

// CC_MakeCredential

#define _UnsealDataAddress (&_UnsealData)
#else
#define _UnsealDataAddress 0
#endif

// CC_Unseal

typedef TPM_RC (Unseal_Entry)(
    Unseal_In                   *in,
    Unseal_Out
);
typedef const struct {
    Unseal_Entry            *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    BYTE                    types[4];
} Unseal_COMMAND_DESCRIPTOR_t;

Unseal_COMMAND_DESCRIPTOR_t _UnsealData = {
    /* entry         */ &TPM2_Unseal,
    /* inSize        */ (UINT16)(sizeof(Unseal_In)),
    /* outSize       */ (UINT16)(sizeof(Unseal_Out)),
    /* offsetOfTypes */ offsetOf(Unseal_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ // No parameter offsets;
    /* types         */ {
        TPM2B_SENSITIVE_DATA_P_MARSHAL,
    END_OF_LIST}
};
#if CC_ObjectChangeAuth
#include "ObjectChangeAuth_fp.h"

typedef TPM_RC (ObjectChangeAuth_Entry)(
    ObjectChangeAuth_In   *in,
    ObjectChangeAuth_Out  *out
);

typedef const struct  {
    ObjectChangeAuth_Entry   *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[2];
    BYTE types[6];
} ObjectChangeAuth_COMMAND_DESCRIPTOR_t;

ObjectChangeAuth_COMMAND_DESCRIPTOR_t _ObjectChangeAuthData = {
    /* entry */       &TPM2_ObjectChangeAuth,
    /* inSize */      (UINT16)(sizeof(ObjectChangeAuth_In)),
    /* outSize */     (UINT16)(sizeof(ObjectChangeAuth_Out)),
    /* offsetOfTypes */ (UINT16)(offsetof(ObjectChangeAuth_COMMAND_DESCRIPTOR_t, types)),
    /* offsets */    {
        (UINT16)(offsetof(ObjectChangeAuth_In, parentHandle)),
        (UINT16)(offsetof(ObjectChangeAuth_In, newAuth))},
    /* types */      {
        TPMI_DH_OBJECT_H_UNMARSHAL,
        TPMI_DH_OBJECT_H_UNMARSHAL,
        TPM2B_AUTH_P_UNMARSHAL,
        END_OF_LIST,
        TPM2B_PRIVATE_P_MARSHAL,
        END_OF_LIST}
};
#endif
// CC_ObjectChangeAuth

#if CC_CreateLoaded
#include "CreateLoaded_fp.h"

typedef TPM_RC (CreateLoaded_Entry)(
    CreateLoaded_In   *in,
    CreateLoaded_Out  *out
);

typedef const struct  {
    CreateLoaded_Entry   *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[5];
    BYTE types[9];
} CreateLoaded_COMMAND_DESCRIPTOR_t;

CreateLoaded_COMMAND_DESCRIPTOR_t _CreateLoadedData = {
    /* entry */       &TPM2_CreateLoaded,
    /* inSize */      (UINT16)(sizeof(CreateLoaded_In)),
    /* outSize */     (UINT16)(sizeof(CreateLoaded_Out)),
    /* offsetOfTypes */ (UINT16)(offsetof(CreateLoaded_COMMAND_DESCRIPTOR_t, types)),
    /* offsets */    {
        (UINT16)(offsetof(CreateLoaded_In, inSensitive)),
        (UINT16)(offsetof(CreateLoaded_In, inPublic)),
        (UINT16)(offsetof(CreateLoaded_Out, outPrivate)),
        (UINT16)(offsetof(CreateLoaded_Out, outPublic)),
        (UINT16)(offsetof(CreateLoaded_Out, name))},
    /* types */      {
        TPMI_DH_PARENT_H_UNMARSHAL + ADD_FLAG,
        TPM2B_SENSITIVE_CREATE_P_UNMARSHAL,
        TPM2B_TEMPLATE_P_UNMARSHAL,
        END_OF_LIST,
        TPM_HANDLE_H_MARSHAL,
        TPM2B_PUBLIC_P_MARSHAL,
        TPM2B_PRIVATE_P_MARSHAL,
        TPM2B_NAME_P_MARSHAL,
        END_OF_LIST}
};
#endif
// CC_CreateLoaded
#define _CreateLoadedDataAddress (&_CreateLoadedData)
#else
#define _CreateLoadedDataAddress 0
#endif

typedef TPM_RC (Duplicate_Entry)(
    Duplicate_In *in,
    Duplicate_Out *out
);

typedef const struct {
    Duplicate_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[5];
    BYTE types[9];
} Duplicate_COMMAND_DESCRIPTOR_t;

Duplicate_COMMAND_DESCRIPTOR_t _DuplicateData = {
    /* entry */ &TPM2_Duplicate,
    /* inSize */ (UINT16)(sizeof(Duplicate_In)),
    /* outSize */ (UINT16)(sizeof(Duplicate_Out)),
    /* offsetOfTypes */ offsetof(Duplicate_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(Duplicate_In, newParentHandle)),
        (UINT16)(offsetof(Duplicate_In, encryptionKeyIn)),
        (UINT16)(offsetof(Duplicate_In, symmetricAlg)),
        (UINT16)(offsetof(Duplicate_Out, duplicate)),
        (UINT16)(offsetof(Duplicate_Out, outSymSeed))},
    /* types */ {
        TPM1_DH_OBJECT_H_UNMARSHAL,
        TPM1_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
        TPM2B_DATA_P_UNMARSHAL,
        TPMT_SYM_DEF_OBJECT_P_UNMARSHAL + ADD_FLAG,
        END_OF_LIST,
        TPM2B_DATA_P_MARSHAL,
        TPM2B_PRIVATE_P_MARSHAL,
        TPM2B_ENCRYPTED_SECRET_P_MARSHAL,
        END_OF_LIST} }
#endif

#define _DuplicateDataAddress (&_DuplicateData)
#else
#define _DuplicateDataAddress 0
#endif

#define _RewrapAddress (&_RewrapData)
#else
#define _RewrapAddress 0
#endif

typedef TPM_RC (Rewrap_Entry)(
    Rewrap_In *in,
    Rewrap_Out *out
);

typedef const struct {
    Rewrap_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[5];
    BYTE types[9];
} Rewrap_COMMAND_DESCRIPTOR_t;

Rewrap_COMMAND_DESCRIPTOR_t _RewrapData = {
    /* entry */ &TPM2_Rewrap,
    /* inSize */ (UINT16)(sizeof(Rewrap_In)),
    /* outSize */ (UINT16)(sizeof(Rewrap_Out)),
    /* offsetOfTypes */ offsetof(Rewrap_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(Rewrap_In, newParent)),
        (UINT16)(offsetof(Rewrap_In, inDuplicate)),
        (UINT16)(offsetof(Rewrap_In, name)),
        (UINT16)(offsetof(Rewrap_In, inSymSeed))},
    /* types */ {
        TPM1_DH_OBJECT_H_UNMARSHAL,
        TPM1_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
        TPM2B_DATA_P_UNMARSHAL,
        TPM2B_DATA_P_MARSHAL,
        TPM2B_PRIVATE_P_MARSHAL,
        TPM2B_ENCRYPTED_SECRET_P_MARSHAL,
        END_OF_LIST} 

/* types */
{TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
 TPM2B_PRIVATE_P_UNMARSHAL,
 TPM2B_PUBLIC_P_UNMARSHAL,
 TPM2B_NAME_P_UNMARSHAL,
 TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL,
 END_OF_LIST,
 TPM2B_PRIVATE_P_MARSHAL,
 TPM2B_PUBLIC_P_MARSHAL,
 TPM2B_ENCRYPTED_SECRET_P_MARSHAL,
 END_OF_LIST}

#define _RewrapDataAddress (&_RewrapData)
#define _ImportDataAddress (&_ImportData)

typedef const struct {
    Import_Entry        *entry,
    UINT16               inSize,
    UINT16               outSize,
    UINT16               offsetOfTypes,
    UINT16               paramOffsets[5];
    BYTE                 types[9];
} Import_COMMAND_DESCRIPTOR_t;

Import_COMMAND_DESCRIPTOR_t _ImportData = {
    /* entry */ &TPM2_Import,
    /* inSize */ (UINT16)(sizeof(Import_In)),
    /* outSize */ (UINT16)(sizeof(Import_Out)),
    /* offsetOfTypes */ offsetof(Import_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(Import_In, encryptionKey)),
        (UINT16)(offsetof(Import_In, objectPublic)),
        (UINT16)(offsetof(Import_In, duplicate)),
        (UINT16)(offsetof(Import_In, inSymSeed)),
        (UINT16)(offsetof(Import_In, symmetricAlg))},
    /* types */
    {TPMI_DH_OBJECT_H_UNMARSHAL,
     TPM2B_DATA_P_UNMARSHAL,
     TPM2B_PUBLIC_P_UNMARSHAL,
     TPM2B_PRIVATE_P_UNMARSHAL,
     TPM2B_ENCRYPTED_SECRET_P_UNMARSHAL,
     TPMT_SYM_DEF_OBJECT_P_UNMARSHAL + ADD_FLAG,
     END_OF_LIST,
     TPM2B_PRIVATE_P_MARSHAL,
     TPM2B_PUBLIC_P_MARSHAL,
     END_OF_LIST}
};

typedef const struct {
    RSA_Encrypt_Entry       *entry,
    UINT16               inSize,
    UINT16               outSize,
    UINT16               offsetOfTypes,
    UINT16               paramOffsets[3];
    BYTE                 types[7];
} RSA_Encrypt_COMMAND_DESCRIPTOR_t;
RSA_Encrypt_COMMAND_DESCRIPTOR_t _RSA_EncryptData = {
    "entry"        &TPM2_RSA_Encrypt,
    "inSize"       (UINT16)(sizeof(RSA_Encrypt_In)),
    "outSize"      (UINT16)(sizeof(RSA_Encrypt_Out)),
    "offsetOfTypes" offsetof(RSA_Encrypt_COMMAND_DESCRIPTOR_t, types),
    "offsets"      {(UINT16)(offsetof(RSA_Encrypt_In, message)),
                    (UINT16)(offsetof(RSA_Encrypt_In, inScheme)),
                    (UINT16)(offsetof(RSA_Encrypt_In, label))},
    "types"        {TPMI_DH_OBJECT_H_UNMARSHAL,
                    TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL,
                    TPMT_RSA_DECRYPT_P_UNMARSHAL + ADD_FLAG,
                    TPM2B_DATA_P_UNMARSHAL,
                    END_OF_LIST,
                    TPM2B_PUBLIC_KEY_RSA_P_MARSHAL,
                    END_OF_LIST}
};
#define _RSA_EncryptDataAddress (&_RSA_EncryptData)
#else
#define _RSA_EncryptDataAddress 0
#endif // CC_RSA_Encrypt

#if CC_RSA_Decrypt
#include "RSA_Decrypt_fp.h"
typedef TPM_RC  (RSA_Decrypt_Entry)(
    RSA_Decrypt_In   *in,
    RSA_Decrypt_Out  *out
);
typedef const struct {
    RSA_Decrypt_Entry   *entry;
    UINT16               inSize;
    UINT16               outSize;
    UINT16               offsetOfTypes;
    UINT16               paramOffsets[3];
    BYTE                  types[7];
} RSA_Decrypt_COMMAND_DESCRIPTOR_t;
RSA_Decrypt_COMMAND_DESCRIPTOR_t _RSA_DecryptData = {
    "entry"        &TPM2_RSA_Decrypt,
    "inSize"       (UINT16)(sizeof(RSA_Decrypt_In)),
    "outSize"      (UINT16)(sizeof(RSA_Decrypt_Out)),
    "offsetOfTypes" offsetof(RSA_Decrypt_COMMAND_DESCRIPTOR_t, types),
    "offsets"      {(UINT16)(offsetof(RSA_Decrypt_In, cipherText)),
                    (UINT16)(offsetof(RSA_Decrypt_In, inScheme)),
                    (UINT16)(offsetof(RSA_Decrypt_In, label))},
    "types"        {TPMI_DH_OBJECT_H_UNMARSHAL,
                    TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL,
                    TPMT_RSA_DECRYPT_P_UNMARSHAL + ADD_FLAG,
                    TPM2B_DATA_P_UNMARSHAL,
                    END_OF_LIST,
                    TPM2B_PUBLIC_KEY_RSA_P_MARSHAL,
                    END_OF_LIST}
};
#define _RSA_DecryptDataAddress (&_RSA_DecryptData)
#else
#define _RSA_DecryptDataAddress 0
#endif // CC_RSA_Decrypt

#if CC_ECDH_KeyGen
#include "ECDH_KeyGen_fp.h"
typedef TPM_RC  (ECDH_KeyGen_Entry)(
    ECDH_KeyGen_In   *in,
    ECDH_KeyGen_Out  *out
);
typedef const struct {
    ECDH_KeyGen_Entry   *entry;
    UINT16               inSize;
    UINT16               outSize;
    UINT16               offsetOfTypes;
} ECDH_KeyGen_COMMAND_DESCRIPTOR_t;
ECDH_KeyGen_COMMAND_DESCRIPTOR_t _ECDH_KeyGenData = {
    "entry"        &TPM2_ECDH_KeyGen,
    "inSize"       (UINT16)(sizeof(ECDH_KeyGen_In)),
    "outSize"      (UINT16)(sizeof(ECDH_KeyGen_Out)),
    "offsetOfTypes" offsetof(ECDH_KeyGen_COMMAND_DESCRIPTOR_t, types),
    "offsets"      {(UINT16)(offsetof(ECDH_KeyGen_In, cipherText)),
                    (UINT16)(offsetof(ECDH_KeyGen_In, inScheme)),
                    (UINT16)(offsetof(ECDH_KeyGen_In, label))},
    "types"        {TPMI_DH_OBJECT_H_UNMARSHAL,
                    TPM2B_PUBLIC_KEY_RSA_P_UNMARSHAL,
                    TPMT_RSA_DECRYPT_P_UNMARSHAL + ADD_FLAG,
                    TPM2B_DATA_P_UNMARSHAL,
                    END_OF_LIST,
                    TPM2B_PUBLIC_KEY_RSA_P_MARSHAL,
                    END_OF_LIST}
};
#define _ECDH_KeyGenDataAddress (&_ECDH_KeyGenData)
#else
#define _ECDH_KeyGenDataAddress 0
#endif // CC_ECDH_KeyGen
BYTE types[5];

} ECDH_KeyGen_COMMAND_DESCRIPTOR_t;

ECDH_KeyGen_COMMAND_DESCRIPTOR_t _ECDH_KeyGenData = {
  /* entry */ &TPM2_ECDH_KeyGen,
  /* inSize */ (UINT16) (sizeof (ECDH_KeyGen_In)),
  /* outSize */ (UINT16) (sizeof (ECDH_KeyGen_Out)),
  /* offsetOfTypes */ offsetOf(ECDH_KeyGen_COMMAND_DESCRIPTOR_t, types),
  /* offsets */ {
    (UINT16) (offsetof(ECDH_KeyGen_Out, pubPoint))},
  /* types */ {
    TPM2B_ECC_POINT_P_MARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
    TPM2B_ECC_POINT_P_UNMARSHAL,
    TPM2B_ECC_POINT_P_PUBLISH,
    TPM2B_ECC_POINT_P_UNPUBLISH,
/* offsets */     // No parameter offsets;
/* types */     {TPMI_ECC_CURVE_P_UNMARSHAL,
END_OF_LIST,  
TPMS_ALGORITHM_DETAIL_ECC_P_MARSHAL,  
END_OF_LIST}

#define _ECC_ParametersDataAddress (&_ECC_ParametersData)
#else
#define _ECC_ParametersDataAddress 0
#endif  // CC_ECC_Parameters

#if CC_ZGen_2Phase
#include "ZGen_2Phase_fp.h"
typedef TPM_RC  (ZGen_2Phase_Entry)(
    ZGen_2Phase_In *in,
    ZGen_2Phase_Out *out
);
typedef const struct 
    {ZGen_2Phase_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offSetOfTypes;
    UINT16 paramOffsets[5];
    BYTE types[9];
} ZGen_2Phase_COMMAND_DESCRIPTOR_t;
ZGen_2Phase_COMMAND_DESCRIPTOR_t _ZGen_2PhaseData = {
/* entry */       &TPM2_ZGen_2Phase,
/* inSize */      (UINT16)(sizeof(ZGen_2Phase_In)),
/* outSize */     (UINT16)(sizeof(ZGen_2Phase_Out)),
/* offSetOfTypes */offsetOf(ZGen_2Phase_COMMAND_DESCRIPTOR_t, types),
/* offsets */     
/* types */       {TPMI_DH_OBJECT_H_UNMARSHAL,
TPM2B_ECC_POINT_P_UNMARSHAL,
TPM2B_ECC_POINT_P_MARSHAL,
END_OF_LIST,  
TPM2B_ECC_POINT_P_MARSHAL,  
TPM2B_ECC_POINT_P_MARSHAL,  
END_OF_LIST}
#endif  // CC_ZGen_2Phase

#if CC_EncryptDecrypt
#include "EncryptDecrypt_fp.h"
typedef TPM_RC  (EncryptDecrypt_Entry)(
    EncryptDecrypt_In *in,
    EncryptDecrypt_Out *out
);
typedef const struct 
    {EncryptDecrypt_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offSetOfTypes;
    UINT16 paramOffsets[5];
    BYTE types[9];
} EncryptDecrypt_COMMAND_DESCRIPTOR_t;
EncryptDecrypt_COMMAND_DESCRIPTOR_t _EncryptDecryptData = {
/* entry */       &TPM2_EncryptDecrypt,
/* inSize */      (UINT16)(sizeof(EncryptDecrypt_In)),
/* outSize */     (UINT16)(sizeof(EncryptDecrypt_Out)),
/* encryption */  {TPM2B_ECC_POINT_P_MARSHAL,  
TPM2B_ECC_POINT_P_MARSHAL,  
TPM2B_ECC_POINT_P_MARSHAL,  
END_OF_LIST,  
TPM2B_ECC_POINT_P_MARSHAL,  
TPM2B_ECC_POINT_P_MARSHAL,  
END_OF_LIST}
typedef const struct {
    EncryptDecrypt2_Entry   *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  inOffsetOfTypes;
    UINT16                  outOffsetOfTypes;
    BYTE                    types[9];
} EncryptDecrypt2_COMMAND_DESCRIPTOR_t;

EncryptDecrypt2_COMMAND_DESCRIPTOR_t _EncryptDecrypt2Data = {
    /* entry */  &TPM2_EncryptDecrypt2,
    /* inSize */  (UINT16)(sizeof(EncryptDecrypt2_In)),
    /* outSize */  (UINT16)(sizeof(EncryptDecrypt2_Out)),
    /* offsetOfTypes */  offsetof(EncryptDecrypt2_COMMAND_DESCRIPTOR_t, types),
    /* offsets */  {
        (UINT16)(offsetof(EncryptDecrypt2_In, inData)),
        (UINT16)(offsetof(EncryptDecrypt2_In, decrypt)),
        (UINT16)(offsetof(EncryptDecrypt2_In, mode)),
        (UINT16)(offsetof(EncryptDecrypt2_In, ivIn)),
        (UINT16)(offsetof(EncryptDecrypt2_Out, ivOut)),
    }
    /* types */  {
        TPMI DH OBJECT_H_UNMARSHAL,
        TPM2B_MAX_BUFFER_P_UNMARSHAL,
        TPMI_YES_NO_P_UNMARSHAL,
        TPM2B_MAX_BUFFER_P_MARSHAL,
        TPM2B_MAX_BUFFER_P_MARSHAL,
        TPM2B_IV_P_MARSHAL,
        TPM2B_MAX_BUFFER_P_MARSHAL,
        TPM2B_MAX_BUFFER_P_MARSHAL,
        END_OF_LIST,
    }
};

#define _EncryptDecrypt2DataAddress (&_EncryptDecrypt2Data)

#endif // CC_EncryptDecrypt2

#ifdef CC_Hash
#include "Hash_fp.h"

typedef TPM_RC  (Hash_Entry)(
    Hash_In                     *in,
    Hash_Out                    *out
);

define const struct {
    Hash_Entry              *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  inOffsetOfTypes;
    UINT16                  outOffsetOfTypes;
    BYTE                    types[9];
} Hash_Entry;

define _EncryptDecrypt2DataAddress (&_EncryptDecrypt2Data)

#endif // CC_Hash
UINT16 inSize;
UINT16 outSize;
UINT16 offsetOfTypes;
UINT16 paramOffsets[3];
BYTE types[7];

} Hash_COMMAND_DESCRIPTOR_t;

Hash_COMMAND_DESCRIPTOR_t _HashData = {
  /* entry */ offsetof(Hash_COMMAND_DESCRIPTOR_t, types),
  /* offsets */ {(UINT16)(offsetof(Hash_In, hashAlg)),
               (UINT16)(offsetof(Hash_In, hierarchy)),
               (UINT16)(offsetof(Hash_Out, validation))},
  /* types */ {TPM2B_MAX_BUFFER_P_UNMARSHAL,
               TPM_ALG_HASH_P_UNMARSHAL,
               TPM_RH_HIERARCHY_P_UNMARSHAL + ADD_FLAG,
               END_OF_LIST,
               TPM2B_DIGEST_P_MARSHAL,
               TPM_TK_HASHCHECK_P_MARSHAL,
               TPM2B_MAX_BUFFER_P_MARSHAL,
               TPM_ALG_HASH_P_MARSHAL,
               TPM_RH_HIERARCHY_P_MARSHAL + ADD_FLAG,
               END_OF_LIST}
};
#define _HashDataAddress (&_HashData)
#else
#define _HashDataAddress 0
#endif // CC_Hash

#endif // CC_HMAC

#include "MAC_fp.h"
typedef TPM_RC (MAC_Entry)(
  MAC_In *in,
  MAC_Out *out)
);

typedef const struct {
  MAC_Entry *entry;
  UINT16 inSize;
  UINT16 outSize;
  UINT16 offsetOfTypes;
  UINT16 paramOffsets[2];
  BYTE types[6];
} MAC_COMMAND_DESCRIPTOR_t;

MAC_COMMAND_DESCRIPTOR_t _MACData = {
  /* entry */ offsetof(MAC_COMMAND_DESCRIPTOR_t, types),
  /* offsets */ {(UINT16)(offsetof(MAC_In, buffer)),
               (UINT16)(offsetof(MAC_In, hashAlg))},
  /* types */ {TPM1_DH_OBJECT_H_UNMARSHAL,
               TPM2B_MAX_BUFFER_P_UNMARSHAL,
               TPM_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
               END_OF_LIST,
               TPM2B_DIGEST_P_MARSHAL,
               TPM2B_MAX_BUFFER_P_MARSHAL,
               TPM_ALG_HASH_P_MARSHAL,
               TPM_RH_HIERARCHY_P_MARSHAL + ADD_FLAG,
               END_OF_LIST}
};
#define _MACDataAddress (&_MACData)
#else
#define _MACDataAddress 0
#endif // CC_MAC

#include "MAC_fp.h"
typedef TPM_RC (MAC_Entry)(
  MAC_In *in,
  MAC_Out *out)
);

typedef const struct {
  MAC_Entry *entry;
1226       UINT16      inSize;
1227       UINT16      outSize;
1228       UINT16      offsetOfTypes;
1229       UINT16      paramOffsets[2];
1230       BYTE        types[6];
1231 } MAC_COMMAND_DESCRIPTOR_t;
1232 MAC_COMMAND_DESCRIPTOR_t _MACData = {
1233     /* entry         */ &TPM2_MAC,
1234     /* inSize        */ (UINT16)(sizeof(MAC_In)),
1235     /* outSize       */ (UINT16)(sizeof(MAC_Out)),
1236     /* offsetOfTypes */ offsetOfMAC_COMMAND_DESCRIPTOR_t, types),
1237     /* offsets       */ {(UINT16)(offsetof(MAC_In, buffer))},
1238     /* types         */ {TPMI_DH_OBJECT_H_UNMARSHAL,
1239                      TPM2B_MAX_BUFFER_P_UNMARSHAL,
1240                      TPM2B_DIGEST_P_MARSHAL,
1241                      TPM2B_DIGEST_P_MARSHAL + ADD_FLAG,
1242                      END_OF_LIST,
1243                      END_OF_LIST}
1244
};
1245 #define _MACDataAddress (&_MACData)
1246 #else
1247 #define _MACDataAddress 0
1248 #endif // CC_MAC
1249
1250 #if CC_GetRandom
1251 #include "GetRandom_fp.h"
1252 typedef TPM_RC  (GetRandom_Entry)(
1253     GetRandom_In *in,
1254     GetRandom_Out *out
1255 );
1256 typedef const struct {
1257     GetRandom_Entry *entry;
1258     UINT16 inSize;
1259     UINT16 outSize;
1260     UINT16 offsetOfTypes;
1261     BYTE types[4];
1262 } GetRandom_COMMAND_DESCRIPTOR_t;
1263 GetRandom_COMMAND_DESCRIPTOR_t _GetRandomData = {
1264     /* entry         */ &TPM2_GetRandom,
1265     /* inSize        */ (UINT16)(sizeof(GetRandom_In)),
1266     /* outSize       */ (UINT16)(sizeof(GetRandom_Out)),
1267     /* offsetOfTypes */ offsetOfGetRandom_COMMAND_DESCRIPTOR_t, types),
1268     /* offsets       */ // No parameter offsets;
1269     /* types         */ {UINT16_P_UNMARSHAL,
1270                      TPM2B_DIGEST_P_MARSHAL,
1271                      TPM2B_DIGEST_P_MARSHAL + ADD_FLAG,
1272                      END_OF_LIST}
1273 );
1274 #define _GetRandomDataAddress (&_GetRandomData)
1275 #else
1276 #define _GetRandomDataAddress 0
1277 #endif // CC_GetRandom
1278
1279 #if CC_StirRandom
1280 #include "StirRandom_fp.h"
1281 typedef TPM_RC  (StirRandom_Entry)(
1282     StirRandom_In *in
1283 );
1284 typedef const struct {
1285     StirRandom_Entry *entry;
1286     UINT16 inSize;
1287     UINT16 outSize;
1288     UINT16 offsetOfTypes;
1289     BYTE types[3];
1290 } StirRandom_COMMAND_DESCRIPTOR_t;
1291 StirRandom_COMMAND_DESCRIPTOR_t _StirRandomData = {
1292     /* entry         */ &TPM2_StirRandom,
/* inSize */
(UINT16)(sizeof(StirRandom_In)),
/* outSize */
0,
/* offsetOfTypes */
offsetof(StirRandom_COMMAND_DESCRIPTOR_t, types),
/* offsets */
// No parameter offsets:
/* types */
{TPM2B_SENSITIVE_DATA_P_UNMARSHAL,
END_OF_LIST,
END_OF_LIST}];
#define _StirRandomDataAddress (&_StirRandomData)
#else
#define _StirRandomDataAddress 0
#endif
// CC_StirRandom

#if CC_HMAC_Start
#include "HMAC_Start_fp.h"
typedef TPM_RC  (HMAC_Start_Entry)(
    HMAC_Start_In               *in,
    HMAC_Start_Out               *out
);
typedef const struct {
    HMAC_Start_Entry        *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[2];
    BYTE                    types[6];
} HMAC_Start_COMMAND_DESCRIPTOR_t;
HMAC_Start_COMMAND_DESCRIPTOR_t _HMAC_StartData = {
    /* entry */
    &TPM2_HMAC_Start,
    /* inSize */
    (UINT16)(sizeof(HMAC_Start_In)),
    /* outSize */
    (UINT16)(sizeof(HMAC_Start_Out)),
    /* offsetOfTypes */
    offsetof(HMAC_Start_COMMAND_DESCRIPTOR_t, types),
    /* offsets */
    {(UINT16)(offsetof(HMAC_Start_In, auth)),
    (UINT16)(offsetof(HMAC_Start_In, hashAlg))},
    /* types */
    {TPMI_DH_OBJECT_H_UNMARSHAL,
    TPM2B_AUTH_P_UNMARSHAL,
    TPM2B_HASH_P_UNMARSHAL + ADD_FLAG,
    END_OF_LIST,
    TPM2_DH_OBJECT_H_MARSHAL,
    END_OF_LIST}
};
#define _HMAC_StartDataAddress (&_HMAC_StartData)
#else
#define _HMAC_StartDataAddress 0
#endif
// CC_HMAC_Start

#if CC_MAC_Start
#include "MAC_Start_fp.h"
typedef TPM_RC  (MAC_Start_Entry)(
    MAC_Start_In               *in,
    MAC_Start_Out               *out
);
typedef const struct {
    MAC_Start_Entry        *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[2];
    BYTE                    types[6];
} MAC_Start_COMMAND_DESCRIPTOR_t;
MAC_Start_COMMAND_DESCRIPTOR_t _MAC_StartData = {
    /* entry */
    &TPM2_MAC_Start,
    /* inSize */
    (UINT16)(sizeof(MAC_Start_In)),
    /* outSize */
    (UINT16)(sizeof(MAC_Start_Out)),
    /* offsetOfTypes */
    offsetof(MAC_Start_COMMAND_DESCRIPTOR_t, types),
    /* offsets */
    {(UINT16)(offsetof(MAC_Start_In, auth)),
    (UINT16)(offsetof(MAC_Start_In, inScheme))},
    /* types */
    {TPMI_DH_OBJECT_H_UNMARSHAL,
typedef TPM_RC (HashSequenceStart_Entry)(
    HashSequenceStart_In *in,
    HashSequenceStart_Out *out
);
typedef const struct {
    HashSequenceStart_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[5];
} HashSequenceStart_COMMAND_DESCRIPTOR_t;
HashSequenceStart_COMMAND_DESCRIPTOR_t _HashSequenceStartData = {
    /* entry */ &TPM2_HashSequenceStart,
    /* inSize */ (UINT16)(sizeof(HashSequenceStart_In)),
    /* outSize */ (UINT16)(sizeof(HashSequenceStart_Out)),
    /* offsetOfTypes */ offsetof(HashSequenceStart_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {(UINT16)(offsetof(HashSequenceStart_In, hashAlg))},
    /* types */ {TPM2B_AUTH_P_UNMARSHAL,

    TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
    TPMI_DH_OBJECT_H_MARSHAL,
    END_OF_LIST,
    TPMI_DH_OBJECT_H_UNMARSHAL,
    TPM2B_MAX_BUFFER_P_UNMARSHAL,
    TPM2B_MAX_BUFFER_P_UNMARSHAL,
    END_OF_LIST,
    END_OF_LIST}
};
#define _HashSequenceStartDataAddress (&_HashSequenceStartData)
}
#define _MAC_StartDataAddress (&_MAC_StartData)
#else
#define _MAC_StartDataAddress 0
#endif
// CC_MAC_Start
#define _MAC_StartDataAddress (&_MAC_StartData)
#else
#define _MAC_StartDataAddress 0
#endif
// CC_MAC_Start
#define _HashSequenceStartDataAddress 0
#endif
// CC_HashSequenceStart
#define _HashSequenceStartDataAddress 0
#endif
// CC_HashSequenceStart
#define _HashSequenceStartDataAddress 0
#endif
// CC_HashSequenceStart

typedef const struct {
    SequenceUpdate_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[4];
} SequenceUpdate_COMMAND_DESCRIPTOR_t;
SequenceUpdate_COMMAND_DESCRIPTOR_t _SequenceUpdateData = {
    /* entry */ &TPM2_SequenceUpdate,
    /* inSize */ (UINT16)(sizeof(SequenceUpdate_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(SequenceUpdate_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {(UINT16)(offsetof(SequenceUpdate_In, buffer))},
    /* types */ {TPM2B_MAX_BUFFER_P_UNMARSHAL,

    TPMI_DH_OBJECT_H_UNMARSHAL,
    TPM2B_MAX_BUFFER_P_UNMARSHAL,
    TPM2B_MAX_BUFFER_P_UNMARSHAL,
    TP
#define _SequenceUpdateDataAddress 0
#endif
// CC_SequenceUpdate

typedef TPM_RC (SequenceComplete_Entry)(
    SequenceComplete_In *in,
    SequenceComplete_Out *out
);

typedef const struct {
    SequenceComplete_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[3];
    BYTE types[7];
} SequenceComplete_COMMAND_DESCRIPTOR_t;

SequenceComplete_COMMAND_DESCRIPTOR_t _SequenceCompleteData = {
    /* entry */ &TPM2_SequenceComplete,
    /* inSize */ (UINT16)(sizeof(SequenceComplete_In)),
    /* outSize */ (UINT16)(sizeof(SequenceComplete_Out)),
    /* offsetOfTypes */ offsetOf(SequenceComplete_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(SequenceCompleteCOMMAND_DESCRIPTOR_t, types)),
        (UINT16)(offsetof(SequenceComplete_In, hierarchy))
    },
    /* types */ {
        TPMI_DH_OBJECT_H_UNMARSHAL,
        TPM2B_MAX_BUFFER_P_UNMARSHAL,
        TPM1_RH_HIERARCHY_P_UNMARSHAL + ADD_FLAG,
        END_OF_LIST,
    }
};

#define _SequenceCompleteDataAddress (&_SequenceCompleteData)
#endif
// CC_SequenceComplete

#if CC_EventSequenceComplete
#include "EventSequenceComplete_fp.h"

typedef TPM_RC (EventSequenceComplete_Entry)(
    EventSequenceComplete_In *in,
    EventSequenceComplete_Out *out
);

typedef const struct {
    EventSequenceComplete_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[2];
    BYTE types[6];
} EventSequenceComplete_COMMAND_DESCRIPTOR_t;

EventSequenceComplete_COMMAND_DESCRIPTOR_t _EventSequenceCompleteData = {
    /* entry */ &TPM2_EventSequenceComplete,
    /* inSize */ (UINT16)(sizeof(EventSequenceComplete_In)),
    /* outSize */ (UINT16)(sizeof(EventSequenceComplete_Out)),
    /* offsetOfTypes */ offsetOf(EventSequenceComplete_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(EventSequenceComplete_In, sequenceHandle)),
        (UINT16)(offsetof(EventSequenceComplete_In, buffer))
    },
    /* types */ {
        TPMI_DH_PCR_H_UNMARSHAL + ADD_FLAG,
        TPM1_RH_HIERARCHY_P_UNMARSHAL,
        TPM2B_MAX_BUFFER_P_UNMARSHAL,
        END_OF_LIST,
        TPML_DIGEST_VALUES_P_MARSHAL,
    }
};

#define _EventSequenceCompleteDataAddress (&_EventSequenceCompleteData)
#endif
// CC_EventSequenceComplete
#define _EventSequenceCompleteDataAddress (&_EventSequenceCompleteData)
#else
#define _EventSequenceCompleteDataAddress 0
#endif

typedef TPM_RC (Certify_Entry) {
    Certify_In *in,
    Certify_Out *out
};
typedef const struct {
    Certify_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[4];
    BYTE types[8];
} Certify_COMMAND_DESCRIPTOR_t;
Certify_COMMAND_DESCRIPTOR_t _CertifyData = {
    /* entry */ &TPM2_Certify,
    /* inSize */ (UINT16)(sizeof(Certify_In)),
    /* outSize */ (UINT16)(sizeof(Certify_Out)),
    /* offsetOfTypes */ offsetof(Certify_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(Certify_In, signHandle)),
        (UINT16)(offsetof(Certify_In, qualifyingData)),
        (UINT16)(offsetof(Certify_In, inScheme)),
        (UINT16)(offsetof(Certify_Out, signature))},
    /* types */ {
        TPMI_DH_OBJECT_H_UNMARSHAL,
        TPM2B_DATA_P_UNMARSHAL + ADD_FLAG,
        TPM2B_ATTEST_P_MARSHAL,
        TPM2B_SIGNATURE_P_MARSHAL,
        END_OF_LIST,
        TPM2B_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
        END_OF_LIST}
};
#define _CertifyDataAddress (&_CertifyData)
#else
#define _CertifyDataAddress 0
#endif

if CC_CertifyCreation
#include "CertifyCreation_fp.h"
typedef TPM_RC (CertifyCreation_Entry) {
    CertifyCreation_In *in,
    CertifyCreation_Out *out
};
typedef const struct {
    CertifyCreation_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[6];
    BYTE types[10];
} CertifyCreation_COMMAND_DESCRIPTOR_t;
CertifyCreation_COMMAND_DESCRIPTOR_t _CertifyCreationData = {
    /* entry */ &TPM2_CertifyCreation,
    /* inSize */ (UINT16)(sizeof(CertifyCreation_In)),
    /* outSize */ (UINT16)(sizeof(CertifyCreation_Out)),
    /* offsetOfTypes */ offsetof(Certify_CREATION_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(CertifyCreation_In, objectHandle)),
        (UINT16)(offsetof(CertifyCreation_In, qualifyingData)),
        (UINT16)(offsetof(CertifyCreation_In, creationHash)),
        (UINT16)(offsetof(CertifyCreation_In, inScheme)),
        (UINT16)(offsetof(CertifyCreation_In, creationTicket)),
1552  (UINT16) {offsetof(CertifyCreation_Out, signature))},
1553  /* types */
1554  {TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
1555  TPM2B_DATA_P_UNMARSHAL,
1556  TPM2B_DIGEST_P_UNMARSHAL,
1557  TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
1558  TPMT_TK_CREATION_P_UNMARSHAL,
1559  END_OF_LIST,
1560  TPM2B_ATTEST_P_MARSHAL,
1561  TPMT_SIGNATURE_P_MARSHAL,
1562  END_OF_LIST}

1563 #define _CertifyCreationDataAddress (&_CertifyCreationData)
1564 #else
1565 #define _CertifyCreationDataAddress 0
1566 #endif // CC_CertifyCreation
1567 #if CC_Quote
1568 #include "Quote_fp.h"
1569 typedef TPM_RC (Quote_Entry)(
1570     Quote_In *in,
1571     Quote_Out *out
1572 );
1573 typedef const struct {
1574     Quote_Entry *entry;
1575     UINT16 inSize;
1576     UINT16 outSize;
1577     UINT16 offsetOfTypes;
1578     UINT16 paramOffsets[4];
1579     BYTE types[8];
1580 } Quote_COMMAND_DESCRIPTOR_t;
1581 Quote_COMMAND_DESCRIPTOR_t _QuoteData = {
1582     /* entry */ &TPM2_Quote,
1583     /* inSize */ (UINT16) {sizeof(Quote_In)},
1584     /* outSize */ (UINT16) {sizeof(Quote_Out)},
1585     /* offsetOfTypes */ offsetof(Quote_COMMAND_DESCRIPTOR_t, types),
1586     /* offsets */ {
1587         (UINT16) {offsetof(Quote_In, qualifyingData)},
1588         (UINT16) {offsetof(Quote_In, inScheme)},
1589         (UINT16) {offsetof(Quote_In, PCRselect)},
1590         (UINT16) {offsetof(Quote_Out, signature))},
1591     /* types */ {
1592         TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
1593         TPM2B_DATA_P_UNMARSHAL,
1594         TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
1595         TPML_PCR_SELECTION_P_UNMARSHAL,
1596         END_OF_LIST,
1597         TPM2B_ATTEST_P_MARSHAL,
1598         TPMT_SIGNATURE_P_MARSHAL,
1599         END_OF_LIST}
1600 #define _QuoteDataAddress (&_QuoteData)
1601 #else
1602 #define _QuoteDataAddress 0
1603 #endif // CC_Quote
1604 #if CC_GetSessionAuditDigest
1605 #include "GetSessionAuditDigest_fp.h"
1606 typedef TPM_RC (GetSessionAuditDigest_Entry)(
1607     GetSessionAuditDigest_In *in,
1608     GetSessionAuditDigest_Out *out
1609 );
1610 typedef const struct {
1611     GetSessionAuditDigest_Entry *entry;
1612     UINT16 inSize;
1613     UINT16 outSize;
1614     UINT16 offsetOfTypes;
1615     UINT16 paramOffsets[5];
1616     BYTE types[9];
1617 } GetSessionAuditDigest_COMMAND_DESCRIPTOR_t;
GetSessionAuditDigest_COMMAND_DESCRIPTOR_t _GetSessionAuditDigestData = {
    /* entry */ &TPM2_GetSessionAuditDigest,
    /* inSize */ (UINT16)(sizeof(GetSessionAuditDigest_In)),
    /* outSize */ (UINT16)(sizeof(GetSessionAuditDigest_Out)),
    /* offsetOfTypes */ offsetof(GetSessionAuditDigest_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {(UINT16)(offsetof(GetSessionAuditDigest_In,
        signHandle)),
        (UINT16)(offsetof(GetSessionAuditDigest_In,
        qualifyingData)),
        (UINT16)(offsetof(GetSessionAuditDigest_In,
        inScheme)),
        (UINT16)(offsetof(GetSessionAuditDigest_Out,
        signature))},
    /* types */ {
        TPMI_RH_ENDORSEMENT_H_UNMARSHAL,
        TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
        TPMI_SH_HMAC_H_UNMARSHAL,
        TPM2B_DATA_P_UNMARSHAL,
        TPM2B_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
        END_OF_LIST,
        TPM2B_ATTEST_P_MARSHAL,
        TPM2B_SIGNATURE_P_MARSHAL,
        END_OF_LIST}
};
#define _GetSessionAuditDigestDataAddress (&_GetSessionAuditDigestData)
#else
#define _GetSessionAuditDigestDataAddress 0
#endif
// CC_GetSessionAuditDigest

typedef TPM_RC (GetCommandAuditDigest_Entry)(
    GetCommandAuditDigest_In *in,
    GetCommandAuditDigest_Out *out)

typedef const struct {
    GetCommandAuditDigest_Entry     *entry;
    UINT16                          inSize;
    UINT16                          outSize;
    UINT16                          offsetOfTypes;
    UINT16                          paramOffsets[4];
    BYTE                            types[8];
} GetCommandAuditDigest_COMMAND_DESCRIPTOR_t;
GetCommandAuditDigest_COMMAND_DESCRIPTOR_t _GetCommandAuditDigestData = {
    /* entry */ &TPM2_GetCommandAuditDigest,
    /* inSize */ (UINT16)(sizeof(GetCommandAuditDigest_In)),
    /* outSize */ (UINT16)(sizeof(GetCommandAuditDigest_Out)),
    /* offsetOfTypes */ offsetof(GetCommandAuditDigest_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {(UINT16)(offsetof(GetCommandAuditDigest_In,
        signHandle)),
        (UINT16)(offsetof(GetCommandAuditDigest_In,
        qualifyingData)),
        (UINT16)(offsetof(GetCommandAuditDigest_In,
        inScheme)),
        (UINT16)(offsetof(GetCommandAuditDigest_Out,
        signature))},
    /* types */ {
        TPMI_RH_ENDORSEMENT_H_UNMARSHAL,
        TPMI_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
        TPM2B_DATA_P_UNMARSHAL,
        TPM2B_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
        END_OF_LIST,
        TPM2B_ATTEST_P_MARSHAL,
        TPM2B_SIGNATURE_P_MARSHAL,
        END_OF_LIST}
typedef TPM_RC (GetTime_Entry)(
  GetTime_In                  *in,
  GetTime_Out                 *out
);

typedef const struct {
  GetTime_Entry *entry;
  UINT16 inSize;
  UINT16 outSize;
  UINT16 offsetOfTypes;
  UINT16 paramOffsets[4];
  BYTE types[8];
} GetTime_COMMAND_DESCRIPTOR_t;

GetTime_COMMAND_DESCRIPTOR_t _GetTimeData = {
  /* entry         */ &TPM2_GetTime,
  /* inSize        */ (UINT16)(sizeof(GetTime_In)),
  /* outSize       */ (UINT16)(sizeof(GetTime_Out)),
  /* offsetOfTypes */ offsetof(GetTime_COMMAND_DESCRIPTOR_t, types),
  /* offsets       */ {
  (UINT16)(offsetof(GetTime_In, signHandle)),
  (UINT16)(offsetof(GetTime_In, qualifyingData)),
  (UINT16)(offsetof(GetTime_In, inScheme)),
  (UINT16)(offsetof(GetTime_Out, signature))},
  /* types         */ {
    TPM1_RH_ENDORSEMENT_H_UNMARSHAL,
    TPM1_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
    TPM2B_DATA_P_UNMARSHAL,
    TPM_SIG_SCHM_P_UNMARSHAL + ADD_FLAG,
    END_OF_LIST,
    TPM2B_ATTEST_P_MARSHAL,
    TPM_SIGNATURE_P_MARSHAL,
    END_OF_LIST
  }
};

#define _GetTimeDataAddress (&_GetTimeData)
#else
#define _GetTimeDataAddress 0
#endif
// CC_GetTime

#define _GetCommandAuditDigestDataAddress (_GetCommandAuditDigestData)
#endif // CC_GetCommandAuditDigest

if CC_GetTime
#include "GetTime_fp.h"

typedef TPM_RC (GetTime_Entry)(
  GetTime_In *in,
  GetTime_Out *out
);

typedef const struct {
  GetTime_Entry *entry;
  UINT16 inSize;
  UINT16 outSize;
  UINT16 offsetOfTypes;
  UINT16 paramOffsets[6];
  BYTE types[10];
} Commit_COMMAND_DESCRIPTOR_t;

Commit_COMMAND_DESCRIPTOR_t _CommitData = {
  /* entry         */ &TPM2_Commit,
  /* inSize        */ (UINT16)(sizeof(Commit_In)),
  /* outSize       */ (UINT16)(sizeof(Commit_Out)),
  /* offsetOfTypes */ offsetof(Commit_COMMAND_DESCRIPTOR_t, types),
  /* offsets       */ {
  (UINT16)(offsetof(Commit_In, P1)),
  (UINT16)(offsetof(Commit_In, s2)),
  (UINT16)(offsetof(Commit_In, y2)),
  (UINT16)(offsetof(Commit_Out, L)),
  (UINT16)(offsetof(Commit_Out, E)),
  (UINT16)(offsetof(Commit_Out, counter))},
}
/* types */
{TPMI_DH_OBJECT_H_UNMARSHAL,
 TPM2B_ECC_POINT_P_UNMARSHAL,
 TPM2B_SENSITIVE_DATA_P_UNMARSHAL,
 TPM2B_ECC_PARAMETER_P_UNMARSHAL,
 END_OF_LIST,
 TPM2B_ECC_POINT_P_MARSHAL,
 TPM2B_ECC_POINT_P_MARSHAL,
 TPM2B_ECC_POINT_P_MARSHAL,
 UINT16_P_MARSHAL,
 END_OF_LIST}

#define _CommitDataAddress (&_CommitData)

#else
#define _CommitDataAddress 0
#endif // CC_Commit

#if CC_EC_Ephemeral
#include "EC_Ephemeral_fp.h"
typedef TPM_RC (EC_Ephemeral_Entry)(
    EC_Ephemeral_In             *in,
    EC_Ephemeral_Out            *out
);

typedef const struct {
    EC_Ephemeral_Entry      *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[1];
    BYTE                    types[5];
} EC_Ephemeral_COMMAND_DESCRIPTOR_t;

EC_Ephemeral_COMMAND_DESCRIPTOR_t _EC_EphemeralData = {
    /* entry */ &TPM2_EC_Ephemeral,
    /* inSize */ (UINT16)(sizeof(EC_Ephemeral_In)),
    /* outSize */ (UINT16)(sizeof(EC_Ephemeral_Out)),
    /* offsetOfTypes */ offsetOf(EC_Ephemeral_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {{(UINT16)(offsetof(EC_Ephemeral_Out, counter))},
    /* types */ {TPMI_ECC_CURVE_P_UNMARSHAL,
    END_OF_LIST,
    TPM2B_ECC_POINT_P_MARSHAL,
    UINT16_P_MARSHAL,
    END_OF_LIST}
};

#define _EC_EphemeralDataAddress (&_EC_EphemeralData)

#else
#define _EC_EphemeralDataAddress 0
#endif // CC_EC_Ephemeral

#if CC_VerifySignature
#include "VerifySignature_fp.h"
typedef TPM_RC (VerifySignature_Entry)(
    VerifySignature_In          *in,
    VerifySignature_Out         *out
);

typedef const struct {
    VerifySignature_Entry   *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[2];
    BYTE                    types[6];
} VerifySignature_COMMAND_DESCRIPTOR_t;

VerifySignature_COMMAND_DESCRIPTOR_t_VerifySignatureData = {
    /* entry */ &TPM2_VerifySignature,
    /* inSize */ (UINT16)(sizeof(VerifySignature_In)),
    /* outSize */ (UINT16)(sizeof(VerifySignature_Out)),
    /* offsetOfTypes */ offsetOf(VerifySignature_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
    (UINT16)(offsetof(VerifySignature_In, digest)),
    (UINT16)(offsetof(VerifySignature_In, signature))},
}
typedef TPM_RC (Sign_Entry)(
   Sign_In *in,
   Sign_Out *out
);

typedef const struct {
   Sign_Entry *entry;
   UINT16 inSize;
   UINT16 outSize;
   UINT16 offsetOfTypes;
   UINT16 paramOffsets[3];
   BYTE types[7];
} Sign_COMMAND_DESCRIPTOR_t;

Sign_COMMAND_DESCRIPTOR_t _SignData = {
   /* entry */ &TPM2_Sign,
   /* inSize */ (UINT16)(sizeof(Sign_In)),
   /* outSize */ (UINT16)(sizeof(Sign_Out)),
   /* offsetOfTypes */ offsetOf(Sign_COMMAND_DESCRIPTOR_t, types),
   /* offsets */ {
      (UINT16)(offsetof(Sign_In, digest)),
      (UINT16)(offsetof(Sign_In, inScheme)),
      (UINT16)(offsetof(Sign_In, validation))},
   /* types */ {
      TPMI_DH_OBJECT_H_UNMARSHAL,
      TPM2B_DIGEST_P_UNMARSHAL,
      TPMT_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
      TPMT_TK_HASHCHECK_P_UNMARSHAL,
      END_OF_LIST,
      TPMT_SIGNATURE_P_MARSHAL,
      END_OF_LIST
   }
};

#define _SignDataAddress (&_SignData)

#define _SignDataAddress (0)

typedef TPM_RC (SetCommandCodeAuditStatus_Entry)(
   SetCommandCodeAuditStatus_In *in
);

SetCommandCodeAuditStatus_In *in

typedef const struct {
   SetCommandCodeAuditStatus_Entry *entry;
   UINT16 inSize;
   UINT16 outSize;
   UINT16 offsetOfTypes;
   UINT16 paramOffsets[3];
   BYTE types[6];
} SetCommandCodeAuditStatus_COMMAND_DESCRIPTOR_t;

SetCommandCodeAuditStatus_COMMAND_DESCRIPTOR_t _SetCommandCodeAuditStatusData = {
   /* entry */ &TPM2_SetCommandCodeAuditStatus,
   /* inSize */ (UINT16)(sizeof(SetCommandCodeAuditStatus_In)),
   /* outSize */ 0,
   /* offsetOfTypes */ offsetOf(SetCommandCodeAuditStatus_COMMAND_DESCRIPTOR_t, types),
1868  /* offsets       */
1869  { (UINT16)(offsetof(SetCommandCodeAuditStatus_In, auditAlg)),
1870      (UINT16)(offsetof(SetCommandCodeAuditStatus_In, setList)),
1871      (UINT16)(offsetof(SetCommandCodeAuditStatus_In, clearList))},
1872  /* types         */
1873  { TPMI_RH_PROVISION_H_UNMARSHAL,
1874      TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
1875      TPML_CC_P_UNMARSHAL,
1876      TPML_CC_P_UNMARSHAL,
1877      END_OF_LIST,
1878      END_OF_LIST}
1879};
1880#define _SetCommandCodeAuditStatusDataAddress (&_SetCommandCodeAuditStatusData)
1881#else
1882#define _SetCommandCodeAuditStatusDataAddress 0
1883#endif
1884// CC_SetCommandCodeAuditStatus
1885#if CC_PCR_Extend
1886#include "PCR_Extend_fp.h"
1887typedef TPM_RC (PCR_Extend_Entry)(
1888    PCR_Extend_In               *in
1889);
1890typedef const struct
1891{ PCR_Extend_Entry        *entry;
1892    UINT16                  inSize;
1893    UINT16                  outSize;
1894    UINT16                  offsetOfTypes;
1895    UINT16                  paramOffsets[1];
1896    BYTE                    types[4];
1897 } PCR_Extend_COMMAND_DESCRIPTOR_t;
1898PCR_Extend_COMMAND_DESCRIPTOR_t _PCR_ExtendData = {
1899  /* entry         */ &TPM2_PCR_Extend,
1900  /* inSize        */ (UINT16)(sizeof(PCR_Extend_In)),
1901  /* outSize       */ 0,
1902  /* offsetOfTypes */  (UINT16)(offsetof(PCR_Extend_COMMAND_DESCRIPTOR_t, types)),
1903  /* offsets       */  { (UINT16)(offsetof(PCR_Extend_In, digests))},
1904  /* types         */  { TPMI_DH_PCR_H_UNMARSHAL + ADD_FLAG,
1905      TPML_DIGEST_VALUES_P_UNMARSHAL,
1906      END_OF_LIST,
1907      END_OF_LIST}
1908};
1909#define _PCR_ExtendDataAddress (&_PCR_ExtendData)
1910#else
1911#define _PCR_ExtendDataAddress 0
1912#endif
1913// CC_PCR_Extend
1914#if CC_PCR_Event
1915#include "PCR_Event_fp.h"
1916typedef TPM_RC (PCR_Event_Entry)(
1917    PCR_Event_In                *in,
1918    PCR_Event_Out               *out
1919);
1920typedef const struct
1921{ PCR_Event_Entry         *entry;
1922    UINT16                  inSize;
1923    UINT16                  outSize;
1924    UINT16                  offsetOfTypes;
1925    UINT16                  paramOffsets[1];
1926    BYTE                    types[5];
1927 } PCR_Event_COMMAND_DESCRIPTOR_t;
1928PCR_EventCOMMAND_DESCRIPTOR_t _PCR_EventData = {
1929  /* entry         */ &TPM2_PCR_Event,
1930  /* inSize        */ (UINT16)(sizeof(PCR_Event_In)),
1931  /* outSize       */ (UINT16)(sizeof(PCR_Event_Out)),
1932  /* offsetOfTypes */  (UINT16)(offsetof(PCR_Event_COMMAND_DESCRIPTOR_t, types)),
1933  /* offsets       */  { (UINT16)(offsetof(PCR_Event_In, eventData))},
1934  /* types         */  { TPMI_DH_PCR_H_UNMARSHAL + ADD_FLAG,
Trusted Platform Module Library

Part 4: Supporting Routines

Page 54

TCG Public Review

Family “2.0”

September 18, 2018

Copyright © TCG 2006-2019

Level 00 Revision 01.50

TYPED PM2B EVENT P UNMARMAL,
END OF LIST,
TPML DIGEST VALUES P MARSHAL,
END OF LIST)

#define _PCR_EventDataAddress (&_PCR_EventData)
#else
#define _PCR_EventDataAddress 0
#endif // CC_PCR_Event
#include "PCR_Read_fp.h"
type def TPM_RC (PCR_Read_Entry) (PCR_Read_In *in,
PCR_Read_Out *out )
);
type def const struct {
    PCR_Read_Entry *entry;
    UINT16 inSize;
    UINT16 * outSize;
    UINT16 * offsetOfTypes;
    UINT16 * paramOffsets[2];
    BYTE types[6];
} PCR_Read_COMMAND_DESCRIPTOR_t;
PCR_Read_COMMAND_DESCRIPTOR_t _PCR_ReadData = {
    /* entry */ &TPM2_PCR_Read,
    /* inSize */ (UINT16)(sizeof (PCR_Read_In)),
    /* outSize */ (UINT16)(sizeof (PCR_Read_Out)),
    /* offsetOfTypes */ offset((PCR_Read_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(PCR_Read_Out, pcrSelectionOut)),
        (UINT16)(offsetof(PCR_Read_Out, pcrValues))},
    /* types */ {TPML_PCR_SELECTION_P_UNMARMAL,
    END OF LIST,
    UINT32_P_MARSHAL,
    TPML_PCR_SELECTION_P_MARSHAL,
    TPML_DIGEST_P_MARSHAL,
    END OF LIST}
};
#define _PCR_ReadDataAddress (&_PCR_ReadData)
#else
#define _PCR_ReadDataAddress 0
#endif // CC_PCR_Read
#include "PCR_Allocate_fp.h"
type def TPM_RC (PCR_Allocate_Entry) (PCR_Allocate_In *in,
PCR_Allocate_Out *out )
);
type def const struct {
    PCR_Allocate_Entry *entry;
    UINT16 inSize;
    UINT16 * outSize;
    UINT16 * offsetOfTypes;
    UINT16 * paramOffsets[4];
    BYTE types[8];
} PCR_Allocate_COMMAND_DESCRIPTOR_t;
PCR_Allocate_COMMAND_DESCRIPTOR_t _PCR_AllocateData = {
    /* entry */ &TPM2_PCR_Allocate,
    /* inSize */ (UINT16)(sizeof (PCR_Allocate_In)),
    /* outSize */ (UINT16)(sizeof (PCR_Allocate_Out)),
    /* offsetOfTypes */ offset((PCR_Allocate_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(PCR_Allocate_Out, pcrAllocation)),
        (UINT16)(offsetof(PCR_Allocate_Out, maxPCR)),
        (UINT16)(offsetof(PCR_Allocate_Out, sizeNeeded)),
        (UINT16)(offsetof(PCR_Allocate_Out, sizeAvailable))},
    /* types */ {TPMI_RH_PLATFORM_H_UNMARMAL,
    END OF LIST,
    TPML_PCR_SELECTION_P_UNMARMAL,
    END OF LIST}];
}


```c
#define _PCR_AllocateDataAddress (_PCR_AllocateData)
#define _PCR_SetAuthPolicyDataAddress (_PCR_SetAuthPolicyData)
```

```c
typedef const struct {
    PCR_SetAuthPolicy_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[3];
    BYTE types[6];
} PCR_SetAuthPolicy_COMMAND_DESCRIPTOR_t;

PCR_SetAuthPolicy_COMMAND_DESCRIPTOR_t _PCR_SetAuthPolicyData = {
    /* entry */ &TPM2_PCR_SetAuthPolicy,
    /* inSize */ (UINT16)(sizeof(PCR_SetAuthPolicy_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(PCR_SetAuthPolicy_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(PCR_SetAuthPolicy_In, authPolicy)),
        (UINT16)(offsetof(PCR_SetAuthPolicy_In, hashAlg)),
        (UINT16)(offsetof(PCR_SetAuthPolicy_In, pcrNum))},
    /* types */ {
        TPMI_RH_PLATFORM_H_UNMARSHAL,
        TPM2B_DIGEST_P_UNMARSHAL,
        TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
        TPMI_DH_PCR_P_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};
```
#define _PCR_SetAuthValueDataAddress (&_PCR_SetAuthValueData)
#else
#define _PCR_SetAuthValueDataAddress 0
#endif // CC_PCR_SetAuthValue

if CC_PCR_Reset
#include "PCR_Reset_fp.h"
typedef TPM_RC (PCR_Reset_Entry)(
    PCR_Reset_In *in
)

typedef const struct {
    PCR_Reset_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    BYTE types[3];
} PCR_Reset_COMMAND_DESCRIPTOR_t;

PCR_Reset_COMMAND_DESCRIPTOR_t _PCR_ResetData = {
    /* entry */ &TPM2_PCR_Reset,
    /* inSize */ (UINT16)(sizeof(PCR_Reset_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(PCR_Reset_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ // No parameter offsets;
    /* types */ {TPMI_DH_PCR_H_UNMARSHAL,
    END_OF_LIST,
    END_OF_LIST}
};
#define _PCR_ResetDataAddress (&_PCR_ResetData)
#else
#define _PCR_ResetDataAddress 0
#endif // CC_PCR_Reset

if CC_PolicySigned
#include "PolicySigned_fp.h"
typedef TPM_RC (PolicySigned_Entry)(
    PolicySigned_In *in,
    PolicySigned_Out *out
)

typedef const struct {
    PolicySigned_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[7];
    BYTE types[11];
} PolicySigned_COMMAND_DESCRIPTOR_t;

PolicySigned_COMMAND_DESCRIPTOR_t _PolicySignedData = {
    /* entry */ &TPM2_PolicySigned,
    /* inSize */ (UINT16)(sizeof(PolicySigned_In)),
    /* outSize */ (UINT16)(sizeof(PolicySigned_Out)),
    /* offsetOfTypes */ offsetof(PolicySigned_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(PolicySigned_In, policySession)),
        (UINT16)(offsetof(PolicySigned_In, nonceTPM)),
        (UINT16)(offsetof(PolicySigned_In, cphashA)),
        (UINT16)(offsetof(PolicySigned_In, policyRef)),
        (UINT16)(offsetof(PolicySigned_In, expiration)),
        (UINT16)(offsetof(PolicySigned_In, auth)),
        (UINT16)(offsetof(PolicySigned_Out, policyTicket))},
    /* types */ {
        TPMI_DH_OBJECT_H_UNMARSHAL,
        TPMI_SH_POLICY_H_UNMARSHAL,
        TPM2B_NONCE_P_UNMARSHAL,
        TPM2B_DIGEST_P_UNMARSHAL,
        TPM2B_NONCE_P_UNMARSHAL,
        INT32_P_UNMARSHAL,
        TPMT_SIGNATURE_P_UNMARSHAL,
        END_OF_LIST,
    --
    END_OF_LIST}
```c
#define _PolicySignedDataAddress (&_PolicySignedData)

#define _PolicySecretDataAddress (&_PolicySecretData)
```

```c
typedef TPM_RC (PolicySecret_Entry)(
    PolicySecret_In *in,
    PolicySecret_Out *out
);
```

```c
typedef const struct {
    PolicySecret_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[6];
    BYTE types[10];
} PolicySecret_COMMAND_DESCRIPTOR_t;
```

```c
PolicySecret_COMMAND_DESCRIPTOR_t _PolicySecretData = {
    /* entry */ &TPM2_PolicySecret,
    /* inSize */ (UINT16)(sizeof(PolicySecret_In)),
    /* outSize */ (UINT16)(sizeof(PolicySecret_Out)),
    /* offsetOfTypes */ offsetof(PolicySecret_COMMAND_DESCRIPTOR_t, types),
    /* offsets */
        (UINT16)(offsetof(PolicySecret_In, policySession)),
        (UINT16)(offsetof(PolicySecret_In, nonceTPM)),
        (UINT16)(offsetof(PolicySecret_In, cpHashA)),
        (UINT16)(offsetof(PolicySecret_In, policyRef)),
        (UINT16)(offsetof(PolicySecret_In, expiration)),
        (UINT16)(offsetof(PolicySecret_Out, policyTicket))},
    /* types */
        TPMI_DH_ENTITY_H_UNMARSHAL,
        TPMI_SH_POLICY_H_UNMARSHAL,
        TPM2B_NONCE_P_UNMARSHAL,
        TPM2B_DIGEST_P_UNMARSHAL,
        TPM2B_NONCE_P_UNMARSHAL,
        INT32_P_UNMARSHAL,
        END_OF_LIST,
        TPM2B_TIMEOUT_P_MARSHAL,
        TPM2B_TIMEOUT_P_MARSHAL,
        TPM2B_TIMEOUT_P_MARSHAL,
        END_OF_LIST}
};
```

```c
#define _PolicySecretDataAddress (&_PolicySecretData)
```

```c
```
```
```
```
```c
typedef TPM_RC (PolicyTicket_Entry)(
    PolicyTicket_In *in,
    PolicyTicket_Out *out
);
```

```c
typedef const struct {
    PolicyTicket_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[5];
    BYTE types[8];
} PolicyTicket_COMMAND_DESCRIPTOR_t;
```

```c
PolicyTicket_COMMAND_DESCRIPTOR_t _PolicyTicketData = {
    /* entry */ &TPM2_PolicyTicket,
    /* inSize */ (UINT16)(sizeof(PolicyTicket_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(PolicyTicket_COMMAND_DESCRIPTOR_t, types),
    /* offsets */
        (UINT16)(offsetof(PolicyTicket_In, policySession)),
        (UINT16)(offsetof(PolicyTicket_In, nonceTPM)),
        (UINT16)(offsetof(PolicyTicket_In, cpHashA)),
        (UINT16)(offsetof(PolicyTicket_In, policyRef)),
        (UINT16)(offsetof(PolicyTicket_In, expiration)),
        (UINT16)(offsetof(PolicyTicket_Out, policyTicket))},
    /* types */
        TPMI_DH_ENTITY_H_UNMARSHAL,
        TPMI_SH_POLICY_H_UNMARSHAL,
        TPM2B_NONCE_P_UNMARSHAL,
        TPM2B_DIGEST_P_UNMARSHAL,
        TPM2B_NONCE_P_UNMARSHAL,
        INT32_P_UNMARSHAL,
        END_OF_LIST,
        TPM2B_TIMEOUT_P_MARSHAL,
        TPM2B_TIMEOUT_P_MARSHAL,
        TPM2B_TIMEOUT_P_MARSHAL,
        END_OF_LIST}
};
```
```
```
/* offsets */
{(UINT16)(offsetof(PolicyTicket_In, timeout)),
(UINT16)(offsetof(PolicyTicket_In, cpHashA)),
(UINT16)(offsetof(PolicyTicket_In, policyRef)),
(UINT16)(offsetof(PolicyTicket_In, authName)),
(UINT16)(offsetof(PolicyTicket_In, ticket))},

/* types */
{TPMI_SH_POLICY_H_UNMARSHAL,
TPM2B_TIMEOUT_P_UNMARSHAL,
TPM2B_DIGEST_P_UNMARSHAL,
TPM2B_NONCE_P_UNMARSHAL,
TPM2B_NAME_P_UNMARSHAL,
TPMT_TK_AUTH_P_UNMARSHAL,
END_OF_LIST,
END_OF_LIST}

#define _PolicyTicketDataAddress (&_PolicyTicketData)
#else
#define _PolicyTicketDataAddress 0
#endif
// CC_PolicyTicket

/* offsets */
{(UINT16)(offsetof(PolicyOR_In, pHashList))},
/* types */
{TPMI_SH_POLICY_H_UNMARSHAL,
TPM2B_DIGEST_P_UNMARSHAL,
END_OF_LIST,
END_OF_LIST}
#define _PolicyORDataAddress (&_PolicyORData)
#else
#define _PolicyORDataAddress 0
#endif
// CC_PolicyOR

/* offsets */
{(UINT16)(offsetof(PolicyPCR_In, pcrDigest)),
(UINT16)(offsetof(PolicyPCR_In, pcrs))},
/* types */
{TPMI_SH_POLICY_H_UNMARSHAL,
 TPM2B_DIGEST_P_UNMARSHAL,
 TPML_PCR_SELECTION_P_UNMARSHAL,
 END_OF_LIST,
 END_OF_LIST}

#define _PolicyPCRDataAddress (&_PolicyPCRData)
#else
#define _PolicyPCRDataAddress 0
#endif // CC_PolicyPCR

#include "PolicyLocality_fp.h"

typedef TPM_RC (PolicyLocality_Entry)(
 PolicyLocality_In *in
);

typedef const struct {
 PolicyLocality_Entry *entry;
 UINT16 inSize;
 UINT16 outSize;
 UINT16 offsetOfTypes;
 UINT16 paramOffsets[1];
 BYTE types[4];
} PolicyLocality_COMMAND_DESCRIPTOR_t _PolicyLocalityData = {
 /* entry */ &TPM2_PolicyLocality,
 /* inSize */ (UINT16)(sizeof(PolicyLocality_In)),
 /* outSize */ 0,
 /* offsetOfTypes */ offset(PolicyLocality_COMMAND_DESCRIPTOR_t, types),
 /* offsets */ {
 (UINT16)(offsetof(PolicyLocality_In, locality)),
 /* types */{
 TPMI_SH_POLICY_H_UNMARSHAL,
 TPMA_LOCALITY_P_UNMARSHAL,
 END_OF_LIST,
 END_OF_LIST}

#define _PolicyLocalityDataAddress (&_PolicyLocalityData)
#else
#define _PolicyLocalityDataAddress 0
#endif // CC_PolicyLocality

#include "PolicyNV_fp.h"

typedef TPM_RC (PolicyNV_Entry)(
 PolicyNV_In *in
);

typedef const struct {
 PolicyNV_Entry *entry;
 UINT16 inSize;
 UINT16 outSize;
 UINT16 offsetOfTypes;
 UINT16 paramOffsets[5];
 BYTE types[8];
} PolicyNV_COMMAND_DESCRIPTOR_t _PolicyNVData = {
 /* entry */ &TPM2_PolicyNV,
 /* inSize */ (UINT16)(sizeof(PolicyNV_In)),
 /* outSize */ 0,
 /* offsetOfTypes */ offset(PolicyNV_COMMAND_DESCRIPTOR_t, types),
 /* offsets */ {
 (UINT16)(offsetof(PolicyNV_In, nvIndex)),
 (UINT16)(offsetof(PolicyNV_In, policySession)),
 (UINT16)(offsetof(PolicyNV_In, operandB)),
 (UINT16)(offsetof(PolicyNV_In, offset)),
 (UINT16)(offsetof(PolicyNV_In, operation))},
 /* types */ {
 TPMI_RH_NV_AUTH_H_UNMARSHAL,
 TPMI_SH_POLICY_H_UNMARSHAL,
 TPM2B_OPERAND_P_UNMARSHAL,
 UINT16_P_UNMARSHAL,

typedef TPM_RC (PolicyCounterTimer_Entry)(
    PolicyCounterTimer_In *in
);
typedef const struct {
    PolicyCounterTimer_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[3];
    BYTE types[6];
} PolicyCounterTimer_COMMAND_DESCRIPTOR_t;
PolicyCounterTimer_COMMAND_DESCRIPTOR_t _PolicyCounterTimerData = {
    /* entry */ &TPM2_PolicyCounterTimer,
    /* inSize */ (UINT16)(sizeof(PolicyCounterTimer_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(PolicyCounterTimer_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(PolicyCounterTimer_In, operandB)),
        (UINT16)(offsetof(PolicyCounterTimer_In, offset)),
        (UINT16)(offsetof(PolicyCounterTimer_In, operation))},
    /* types */ {
        TPMI_SH_POLICY_H_UNMARSHAL,
        TPM2B_OPERAND_P_UNMARSHAL,
        UINT16_P_UNMARSHAL,
        TPM_EO_P_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};
#define _PolicyCounterTimerDataAddress (&_PolicyCounterTimerData)
#endif // CC_PolicyCounterTimer

#if CC_PolicyCommandCode
#include "PolicyCommandCode_fp.h"
typedef TPM_RC (PolicyCommandCode_Entry)(
    PolicyCommandCode_In *in
);
typedef const struct {
    PolicyCommandCode_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[4];
} PolicyCommandCode_COMMAND_DESCRIPTOR_t;
PolicyCommandCode_COMMAND_DESCRIPTOR_t _PolicyCommandCodeData = {
    /* entry */ &TPM2_PolicyCommandCode,
    /* inSize */ (UINT16)(sizeof(PolicyCommandCode_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(PolicyCommandCode_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(PolicyCommandCode_In, code))},
    /* types */ {
        TPMI_SH_POLICY_H_UNMARSHAL,
        TPM_CC_P_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};
#define _PolicyCommandCodeDataAddress (&_PolicyCommandCodeData)
#else
#define _PolicyCommandCodeDataAddress 0
#endif // CC_PolicyCommandCode
2389 #define _PolicyCommandCodeDataAddress (&_PolicyCommandCodeData)
2390 #else
2391 #define _PolicyCommandCodeDataAddress 0
2392 #endif // CC_PolicyCommandCode
2393 if CC_PolicyPhysicalPresence
2394 #include "PolicyPhysicalPresence_fp.h"
2395 typedef TPM_RC (PolicyPhysicalPresence_Entry)(
2396 PolicyPhysicalPresence_In *in
2397 }
2398 );
2399 typedef const struct {
2400 PolicyPhysicalPresence_Entry *entry;
2401 UINT16 inSize;
2402 UINT16 outSize;
2403 UINT16 offsetOfTypes;
2404 BYTE types[3];
2405 } PolicyPhysicalPresence_COMMAND_DESCRIPTOR_t;
2406 PolicyPhysicalPresence_COMMAND_DESCRIPTOR_t _PolicyPhysicalPresenceData = {
2407 /* entry */ &TPM2_PolicyPhysicalPresence,
2408 /* inSize */ (UINT16)(sizeof(PolicyPhysicalPresence_In)),
2409 /* outSize */ 0,
2410 /* offsetOfTypes */
2411 offsetOf(PolicyPhysicalPresence_COMMAND_DESCRIPTOR_t, types),
2412 /* offsets */
2413 // No parameter offsets:
2414 /* types */
2415 {TPMI_SH_POLICY_H_UNMARSHAL,
2416 END_OF_LIST,
2417 END_OF_LIST}
2418 }
2419 #define _PolicyPhysicalPresenceDataAddress (&_PolicyPhysicalPresenceData)
2420 #else
2421 #define _PolicyPhysicalPresenceDataAddress 0
2422 #endif // CC_PolicyPhysicalPresence
2423 if CC_PolicyCpHash
2424 #include "PolicyCpHash_fp.h"
2425 typedef TPM_RC (PolicyCpHash_Entry)(
2426 PolicyCpHash_In *in
2427 }
2428 );
2429 typedef const struct {
2430 PolicyCpHash_Entry *entry;
2431 UINT16 inSize;
2432 UINT16 outSize;
2433 UINT16 offsetOfTypes;
2434 UINT16 paramOffsets[1];
2435 BYTE types[4];
2436 } PolicyCpHash_COMMAND_DESCRIPTOR_t;
2437 PolicyCpHash_COMMAND_DESCRIPTOR_t _PolicyCpHashData = {
2438 /* entry */ &TPM2_PolicyCpHash,
2439 /* inSize */ (UINT16)(sizeof(PolicyCpHash_In)),
2440 /* outSize */ 0,
2441 /* offsetOfTypes */
2442 offsetOf(PolicyCpHash_COMMAND_DESCRIPTOR_t, types),
2443 /* offsets */
2444 {(UINT16)(offsetof(PolicyCpHash_In, cpHashA))},
2445 /* types */
2446 {TPMI_SH_POLICY_H_UNMARSHAL,
2447 TPM2B_DIGEST_P_UNMARSHAL,
2448 END_OF_LIST,
2449 END_OF_LIST}
2450 #define _PolicyCpHashDataAddress (&_PolicyCpHashData)
2451 #else
2452 #define _PolicyCpHashDataAddress 0
2453 #endif // CC_PolicyCpHash
2454 if CC_PolicyNameHash
2455 #include "PolicyNameHash_fp.h"
2456 typedef TPM_RC (PolicyNameHash_Entry)(
2457 PolicyNameHash_In *in
2458 );
2459 typedef const struct {
2460 PolicyNameHash_Entry *entry;
2461 UINT16 inSize;
2462 UINT16 outSize;
2463 UINT16 offsetOfTypes;
2464 UINT16 paramOffsets[1];
2465 BYTE types[4];
2466 } PolicyNameHash_COMMAND_DESCRIPTOR_t;
2467 PolicyNameHash_COMMAND_DESCRIPTOR_t _PolicyNameHashData = {
2468 /* entry */ &TPM2_PolicyNameHash,
2469 /* inSize */ (UINT16)(sizeof(PolicyNameHash_In)),
2470 /* outSize */ 0,
2471 /* offsetOfTypes */
2472 offsetOf(PolicyNameHash_COMMAND_DESCRIPTOR_t, types),
2473 /* offsets */
2474 {(UINT16)(offsetof(PolicyNameHash_In, cpHashA))},
2475 /* types */
2476 {TPMI_SH_POLICY_H_UNMARSHAL,
2477 TPM2B_DIGEST_P_UNMARSHAL,
2478 END_OF_LIST,
2479 END_OF_LIST}
2480 #define _PolicyNameHashDataAddress (&_PolicyNameHashData)
2481 #else
2482 #define _PolicyNameHashDataAddress 0
2483 #endif // CC_PolicyNameHash
PolicyNameHash_ENTRY *entry;
UINT16 inSize;
UINT16 outSize;
UINT16 offsetOfTypes;
UINT16 paramOffsets[1];
BYTE types[4];
}
PolicyNameHash_COMMAND_DESCRIPTOR_t _PolicyNameHashData = {
/* entry */ &TPM2_PolicyNameHash,
/* inSize */ (UINT16)(sizeof(PolicyNameHash_In)),
/* outSize */ 0,
/* offsetOfTypes */ offsetOf(PolicyNameHash_COMMAND_DESCRIPTOR_t, types),
/* offsets */ {(UINT16)(offsetof(PolicyNameHash_In, nameHash))},
/* types */ {TPMI_SH_POLICY_H_UNMARSHAL,
TPM2B_DIGEST_P_UNMARSHAL,
END_OF_LIST,
END_OF_LIST};
#define _PolicyNameHashDataAddress (&_PolicyNameHashData)
#else
#define _PolicyNameHashDataAddress 0
#endif // CC_PolicyNameHash
#if CC_PolicyDuplicationSelect
#include "PolicyDuplicationSelect_fp.h"
typedef TPM_RC (PolicyDuplicationSelect_Entry)(
    PolicyDuplicationSelect_In *in
    )
PolicyDuplicationSelect_In;
}
PolicyDuplicationSelect_COMMAND_DESCRIPTOR_t _PolicyDuplicationSelectData = {
/* entry */ &TPM2_PolicyDuplicationSelect,
/* inSize */ (UINT16)(sizeof(PolicyDuplicationSelect_In)),
/* outSize */ 0,
/* offsetOfTypes */ offsetOf(PolicyDuplicationSelect_COMMAND_DESCRIPTOR_t, types),
/* offsets */ {(UINT16)(offsetof(PolicyDuplicationSelect_In, objectName))},
/* types */ {TPMI_SH_POLICY_H_UNMARSHAL,
TPM2B_NAME_P_UNMARSHAL,
TPM2B_NAME_P_UNMARSHAL,
TPMI_YES_NO_P_UNMARSHAL,
END_OF_LIST,
END_OF_LIST};
#define _PolicyDuplicationSelectDataAddress (&_PolicyDuplicationSelectData)
#else
#define _PolicyDuplicationSelectDataAddress 0
#endif // CC_PolicyDuplicationSelect
#if CC_PolicyAuthorize
#include "PolicyAuthorize_fp.h"
typedef TPM_RC (PolicyAuthorize_Entry)(
    PolicyAuthorize_In *in
    )
PolicyAuthorize_In;
};
PolicyAuthorize_ENTRY *entry;
UINT16 inSize;
Part 4: Supporting Routines

Trusted Platform Module Library

2516       UINT16          outSize;
2517       UINT16          offsetOfTypes;
2518       UINT16          paramOffsets[4];
2519       BYTE            types[7];
2520 } PolicyAuthorize_COMMAND_DESCRIPTOR_t;
2521 PolicyAuthorize_COMMAND_DESCRIPTOR_t _PolicyAuthorizeData = {
2522   /* entry */   &TPM2_PolicyAuthorize,
2523   /* inSize */   (UINT16)(sizeof(PolicyAuthorize_In)),
2524   /* outSize */  0,
2525   /* offsetOfTypes */  offsetof(PolicyAuthorize_COMMAND_DESCRIPTOR_t, types),
2526   /* offsets */  
2527   /* types */   {TPMI_SH_POLICY_H_UNMARSHAL,
2528                  TPM2B_DIGEST_P_UNMARSHAL,
2529                  TPM2B_NONCE_P_UNMARSHAL,
2530                  TPM2B_NAME_P_UNMARSHAL,
2531                  TPM_TK_VERIFIED_P_UNMARSHAL,
2532                  END_OF_LIST,
2533                  END_OF_LIST}};
2534 #define _PolicyAuthorizeDataAddress (_PolicyAuthorizeData)
2535 #else
2536 #define _PolicyAuthorizeDataAddress 0
2537 #endif // CC_PolicyAuthorize
2538 #if CC_PolicyAuthValue
2539 #include "PolicyAuthValue_fp.h"
2540 typedef TPM_RC  (PolicyAuthValue_Entry)(
2541    PolicyAuthValue_In *in
2542 );
2543 typedef const struct {
2544    PolicyAuthValue_Entry   *entry;
2545    UINT16                  inSize;
2546    UINT16                  outSize;
2547    UINT16                  offsetOfTypes;
2548    BYTE                    types[3];
2549 } PolicyAuthValue_COMMAND_DESCRIPTOR_t;
2550 PolicyAuthValue_COMMAND_DESCRIPTOR_t _PolicyAuthValueData = {
2551   /* entry */   &TPM2_PolicyAuthValue,
2552   /* inSize */   (UINT16)(sizeof(PolicyAuthValue_In)),
2553   /* offsetOfTypes */  offsetof(PolicyAuthValue_COMMAND_DESCRIPTOR_t, types),
2554   /* types */   {TPMI_SH_POLICY_H_UNMARSHAL,
2555                  END_OF_LIST,
2556                  END_OF_LIST}];
2557 #define _PolicyAuthValueDataAddress (_PolicyAuthValueData)
2558 #else
2559 #define _PolicyAuthValueDataAddress 0
2560 #endif // CC_PolicyAuthValue
2561 #if CC_PolicyPassword
2562 #include "PolicyPassword_fp.h"
2563 typedef TPM_RC  (PolicyPassword_Entry)(
2564    PolicyPassword_In *in
2565 );
2566 typedef const struct {
2567    PolicyPassword_Entry    *entry;
2568    UINT16                  inSize;
2569    UINT16                  outSize;
2570    UINT16                  offsetOfTypes;
2571    BYTE                    types[3];
2572 } PolicyPassword_COMMAND_DESCRIPTOR_t;
2573 PolicyPassword_COMMAND_DESCRIPTOR_t _PolicyPasswordData = {
2574   /* entry */   &TPM2_PolicyPassword,
2582 /* inSize */ (UINT16),sizeof(PolicyPassword_In)),
2583 /* outSize */ 0,
2584 /* offsetOfTypes */ offsetof(PolicyPassword_COMMAND_DESCRIPTOR_t, types),
2585 /* offsets */ // No parameter offsets;
2586 /* types */ {TPMI_SH_POLICY_H_UNMARSHAL,
2587 END_OF_LIST,
2588 END_OF_LIST}
2589 );
2590 #define _PolicyPasswordDataAddress (&_PolicyPasswordData)
2591 #else
2592 #define _PolicyPasswordDataAddress 0
2593 #endif // CC_PolicyPassword
2594 #if CC_PolicyGetDigest
2595 #include "PolicyGetDigest_fp.h"
2596 typedef TPM_RC (PolicyGetDigest_Entry)(
2597 PolicyGetDigest_In *in,
2598 PolicyGetDigest_Out *out)
2599 );
2600 typedef const struct {
2601 PolicyGetDigest_Entry *entry;
2602 UINT16 inSize;
2603 UINT16 outSize;
2604 UINT16 offsetOfTypes;
2605 BYTE types[4];
2606 } PolicyGetDigest_COMMAND_DESCRIPTOR_t;
2607 PolicyGetDigest_COMMAND_DESCRIPTOR_t _PolicyGetDigestData = {
2608 /* entry */ &TPM2_PolicyGetDigest,
2609 /* inSize */ (UINT16)(sizeof(PolicyGetDigest_In)),
2610 /* outSize */ (UINT16)(sizeof(PolicyGetDigest_Out)),
2611 /* offsetOfTypes */ offsetof(PolicyGetDigest_COMMAND_DESCRIPTOR_t, types),
2612 /* offsets */ // No parameter offsets;
2613 /* types */ {TPMI_SH_POLICY_H_UNMARSHAL,
2614 END_OF_LIST,
2615 TPM2B_DIGEST_P_MARSHAL,
2616 END_OF_LIST}
2617 );
2618 #define _PolicyGetDigestDataAddress (&_PolicyGetDigestData)
2619 #else
2620 #define _PolicyGetDigestDataAddress 0
2621 #endif // CC_PolicyGetDigest
2622 #if CC_PolicyNvWritten
2623 #include "PolicyNvWritten_fp.h"
2624 typedef TPM_RC (PolicyNvWritten_Entry)(
2625 PolicyNvWritten_In *in,
2626 PolicyNvWritten_Out *out)
2627 );
2628 typedef const struct {
2629 PolicyNvWritten_Entry *entry;
2630 UINT16 inSize;
2631 UINT16 outSize;
2632 UINT16 offsetOfTypes;
2633 UINT16 paramOffsets[1];
2634 BYTE types[4];
2635 } PolicyNvWritten_COMMAND_DESCRIPTOR_t;
2636 PolicyNvWritten_COMMAND_DESCRIPTOR_t _PolicyNvWrittenData = {
2637 /* entry */ &TPM2_PolicyNvWritten,
2638 /* inSize */ (UINT16)(sizeof(PolicyNvWritten_In)),
2639 /* outSize */ 0,
2640 /* offsetOfTypes */ offsetof(PolicyNvWritten_COMMAND_DESCRIPTOR_t, types),
2641 /* offsets */ {(UINT16)(offsetof(PolicyNvWritten_In, writtenSet))},
2642 /* types */ {TPMI_SH_POLICY_H_UNMARSHAL,
2643 TPM_YES_NO_P_UNMARSHAL,
2644 END_OF_LIST,
2645 END_OF_LIST}
2646 );
2647 #define _PolicyNvWrittenDataAddress (&_PolicyNvWrittenData)
2648 #else
2649 #define _PolicyNvWrittenDataAddress 0
2650 #endif // CC_PolicyNvWritten
2651
#define _PolicyNvWrittenDataAddress 0
#endif  // CC_PolicyNvWritten
#endif  // CC_PolicyTemplate

typedef TPM_RC (PolicyTemplate_Entry)(
    PolicyTemplate_In *in
);

typedef const struct {
    PolicyTemplate_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[4];
} PolicyTemplate_COMMAND_DESCRIPTOR_t;

PolicyTemplate_COMMAND_DESCRIPTOR_t _PolicyTemplateData = {
    /* entry         */ &TPM2_PolicyTemplate,
    /* inSize        */ (UINT16)(sizeof(PolicyTemplate_In)),
    /* outSize       */ 0,
    /* offsetOfTypes */ offset(PolicyTemplate_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {{(UINT16)(offsetof(PolicyTemplate_In, templateHash))},
    /* types         */ {{TPMI_SH_POLICY_H_UNMARSHAL,
    TPM2B_DIGEST_P_UNMARSHAL,
    END_OF_LIST,}}
};
#define _PolicyTemplateDataAddress (&_PolicyTemplateData)
#endif  // CC_PolicyTemplate

#if CC_PolicyAuthorizeNV
#include "PolicyAuthorizeNV_fp.h"

typedef TPM_RC (PolicyAuthorizeNV_Entry)(
    PolicyAuthorizeNV_In *in
);

typedef const struct {
    PolicyAuthorizeNV_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[2];
    BYTE types[5];
} PolicyAuthorizeNV_COMMAND_DESCRIPTOR_t;

PolicyAuthorizeNV_COMMAND_DESCRIPTOR_t _PolicyAuthorizeNVData = {
    /* entry         */ &TPM2_PolicyAuthorizeNV,
    /* inSize        */ (UINT16)(sizeof(PolicyAuthorizeNV_In)),
    /* outSize       */ 0,
    /* offsetOfTypes */ offset(PolicyAuthorizeNV_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {{(UINT16)(offsetof(PolicyAuthorizeNV_In, nvIndex))},
    (UINT16)(offsetof(PolicyAuthorizeNV_In, policySession))},
    /* types         */ {{TPMI_RH_NV_AUTH_H_UNMARSHAL,
    TPMI_RH_NV_INDEX_H_UNMARSHAL,
    TPMI_SH_POLICY_H_UNMARSHAL,
    END_OF_LIST,}}
};
#define _PolicyAuthorizeNVDataAddress (&_PolicyAuthorizeNVData)
#endif  // CC_PolicyAuthorizeNV

#if CC_CreatePrimary
#include "CreatePrimary_fp.h"

typedef TPM_RC (CreatePrimary_Entry)(


typedef const struct {
    CreatePrimary_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[9];
    BYTE types[13];
} CreatePrimary_COMMAND_DESCRIPTOR_t;

CreatePrimary_COMMAND_DESCRIPTOR_t _CreatePrimaryData = {
    /* entry */ &TPM2_CreatePrimary,
    /* inSize */ (UINT16)(sizeof(CreatePrimary_In)),
    /* outSize */ (UINT16)(sizeof(CreatePrimary_Out)),
    /* offsetOfTypes */ offsetof(CreatePrimary_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(CreatePrimary_In, inSensitive)),
        (UINT16)(offsetof(CreatePrimary_In, inPublic)),
        (UINT16)(offsetof(CreatePrimary_In, outsideInfo)),
        (UINT16)(offsetof(CreatePrimary_In, creationPCR)),
        (UINT16)(offsetof(CreatePrimary_Out, outPublic)),
        (UINT16)(offsetof(CreatePrimary_Out, creationData)),
        (UINT16)(offsetof(CreatePrimary_Out, creationHash)),
        (UINT16)(offsetof(CreatePrimary_Out, creationTicket)),
        (UINT16)(offsetof(CreatePrimary_Out, name))},
    /* types */ {
        TPMI_RH_HIERARCHY_H_UNMARSHAL + ADD_FLAG,
        TPM2B_SENSITIVE CREATE_P_UNMARSHAL,
        TPM2B_PUBLIC_P_UNMARSHAL,
        TPM2B_DATA_P_UNMARSHAL,
        TPML_PCR_SELECTION_P_UNMARSHAL,
        END_OF_LIST,
        TPM_HANDLE_H_MARSHAL,
        TPM2B_PUBLIC_P_MARSHAL,
        TPM2B_CREATION_DATA_P_MARSHAL,
        TPM2B_DIGEST_P_MARSHAL,
        TPMT_TK_CREATION_P_MARSHAL,
        TPM2B_NAME_P_MARSHAL,
        END_OF_LIST}
};

#define _CreatePrimaryDataAddress (&_CreatePrimaryData)
#else
#define _CreatePrimaryDataAddress 0
#endif // CC_CreatePrimary

#define CC_HierarchyControl
#include "HierarchyControl_fp.h"

typedef TPM_RC (HierarchyControl_Entry)(
    HierarchyControl_In *in
);

typedef const struct {
    HierarchyControl_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[2];
    BYTE types[5];
} HierarchyControl_COMMAND_DESCRIPTOR_t;

HierarchyControl_COMMAND_DESCRIPTOR_t _HierarchyControlData = {
    /* entry */ &TPM2_HierarchyControl,
    /* inSize */ (UINT16)(sizeof(HierarchyControl_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(HierarchyControl_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(HierarchyControl_In, enable)),
        (UINT16)(offsetof(HierarchyControl_In, state))},
    /* types */ {
        TPMI_RH_HIERARCHY_H_UNMARSHAL,
        TPMI_RH_ENABLES_P_UNMARSHAL,
        TPMI_YES_NO_P_UNMARSHAL,
        END_OF_LIST, -
2780 #define _HierarchyControlDataAddress (&_HierarchyControlData)
2781 #else
2782 #define _HierarchyControlDataAddress 0
2783 endif // CC_HierarchyControl
2784 #if CC_SetPrimaryPolicy
2785 #include "SetPrimaryPolicy_fp.h"
2786 typedef TPM_RC (SetPrimaryPolicy_Entry)(
2787 SetPrimaryPolicy_In *in
2788 );
2789 typedef const struct {
2790 SetPrimaryPolicy_Entry *entry;
2791 UINT16 inSize;
2792 UINT16 outSize;
2793 UINT16 offsetOfTypes;
2794 UINT16 paramOffsets[2];
2795 BYTE types[5];
2796 } SetPrimaryPolicy_COMMAND_DESCRIPTOR_t;
2797 SetPrimaryPolicy_COMMAND_DESCRIPTOR_t _SetPrimaryPolicyData = {
2798 /* entry */ &TPM2_SetPrimaryPolicy,
2799 /* inSize */ (UINT16)(sizeof(SetPrimaryPolicy_In)),
2800 /* outSize */ 0,
2801 /* offsetOfTypes */ offsetOf(SetPrimaryPolicy_COMMAND_DESCRIPTOR_t, types),
2802 /* offsets */ {
2803 (UINT16)(offsetof(SetPrimaryPolicy_In, authPolicy)),
2804 (UINT16)(offsetof(SetPrimaryPolicy_In, hashAlg))},
2805 /* types */ {
2806 TPMI_RH_HIERARCHY_AUTH_H_UNMARSHAL,
2807 TPM2B_DIGEST_P_UNMARSHAL,
2808 TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
2809 END_OF_LIST,
2810 END_OF_LIST}
2811
2812 #define _SetPrimaryPolicyDataAddress (&_SetPrimaryPolicyData)
2813 #else
2814 #define _SetPrimaryPolicyDataAddress 0
2815 #endif // CC_SetPrimaryPolicy
2816 #if CC_ChangePPS
2817 #include "ChangePPS_fp.h"
2818 typedef TPM_RC (ChangePPS_Entry)(
2819 ChangePPS_In *in
2820 );
2821 typedef const struct {
2822 ChangePPS_Entry *entry;
2823 UINT16 inSize;
2824 UINT16 outSize;
2825 UINT16 offsetOfTypes;
2826 BYTE types[3];
2827 } ChangePPS_COMMAND_DESCRIPTOR_t;
2828 ChangePPS_COMMAND_DESCRIPTOR_t _ChangePPSData = {
2829 /* entry */ &TPM2_ChangePPS,
2830 /* inSize */ (UINT16)(sizeof(ChangePPS_In)),
2831 /* outSize */ 0,
2832 /* offsetOfTypes */ offsetOf(ChangePPS_COMMAND_DESCRIPTOR_t, types),
2833 /* offsets */ // No parameter offsets;
2834 /* types */ {
2835 TPMI_RH_PLATFORM_H_UNMARSHAL,
2836 TPM2B_DIGEST_P_UNMARSHAL + ADD_FLAG,
2837 TPMI_ALG_HASH_P_UNMARSHAL + ADD_FLAG,
2838 END_OF_LIST,
2839 END_OF_LIST}
2840
2841 #define _ChangePPSDataAddress (&_ChangePPSData)
2842 #else
2843 #define _ChangePPSDataAddress 0
2844 #endif // CC_ChangePPS
2845 #if CC_ChangeEPS
2846 #include "ChangeEPS_fp.h"
2847 typedef TPM_RC (ChangeEPS_Entry)(
2848 ChangeEPS_In *in
2849 );
typedef const struct {
    ChangeEPS_Entry *entry;
    UINT16         inSize;
    UINT16         outSize;
    UINT16         offsetOfTypes;
    BYTE           types[3];
} ChangeEPS_COMMAND_DESCRIPTOR_t;
ChangeEPS_COMMAND_DESCRIPTOR_t _ChangeEPSData = {
    /* entry */ &TPM2_ChangeEPS,
    /* inSize */ (UINT16)(sizeof(ChangeEPS_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(ChangeEPS_COMMAND_DESCRIPTOR_t, types),
    /* offsets */   // No parameter offsets:
    /* types */    {TPMI_RH_PLATFORM_H_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};
#define _ChangeEPSDataAddress (&_ChangeEPSData)
else
#define _ChangeEPSDataAddress 0
#endif // CC_ChangeEPS
#endif // CC_Clear
#include "Clear_fp.h"
typedef TPM_RC (Clear_Entry)(
    Clear_In  *in )
);
typedef const struct {
    Clear_Entry      *entry;
    UINT16         inSize;
    UINT16         outSize;
    UINT16         offsetOfTypes;
    BYTE           types[3];
} Clear_COMMAND_DESCRIPTOR_t;
Clear_COMMAND_DESCRIPTOR_t _ClearData = {
    /* entry */ &TPM2_Clear,
    /* inSize */ (UINT16)(sizeof(Clear_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(Clear_COMMAND_DESCRIPTOR_t, types),
    /* offsets */   // No parameter offsets:
    /* types */    {TPMI_RH_CLEAR_H_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};
#define _ClearDataAddress (&_ClearData)
else
#define _ClearDataAddress 0
#endif // CC_Clear
#include "ClearControl_fp.h"
typedef TPM_RC (ClearControl_Entry)(
    ClearControl_In  *in )
);
typedef const struct {
    ClearControl_Entry *entry;
    UINT16         inSize;
    UINT16         outSize;
    UINT16         offsetOfTypes;
    BYTE           types[4];
} ClearControl_COMMAND_DESCRIPTOR_t;
ClearControl_COMMAND_DESCRIPTOR_t _ClearControlData = {
    /* entry */ &TPM2_ClearControl,
    /* inSize */ (UINT16)(sizeof(ClearControl_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(ClearControl_COMMAND_DESCRIPTOR_t, types),
/* offsets */
{(UINT16)(offsetof(ClearControl_In, disable))},
/* types */
{TPMI_RH_CLEAR_H_UNMARSHAL,
TPMI_YES_NO_P_UNMARSHAL,
END_OF_LIST,
END_OF_LIST}
};
#define _ClearControlDataAddress (&_ClearControlData)
#else
#define _ClearControlDataAddress 0
#endif
// CC_ClearControl
#if CC_HierarchyChangeAuth
#include "HierarchyChangeAuth_fp.h"
typedef TPM_RC (HierarchyChangeAuth_Entry)(
    HierarchyChangeAuth_In *in
);
typedef const struct {
    HierarchyChangeAuth_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[4];
} HierarchyChangeAuth_COMMAND_DESCRIPTOR_t;
HierarchyChangeAuth_COMMAND_DESCRIPTOR_t _HierarchyChangeAuthData = {
    /* entry */ &TPM2_HierarchyChangeAuth,
    /* inSize */ (UINT16)(sizeof(HierarchyChangeAuth_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offset(HierarchyChangeAuth_COMMAND_DESCRIPTOR_t, types),
    /* offsets */
{(UINT16)(offsetof(HierarchyChangeAuth_In, newAuth))},
    /* types */
{TPMI_RH_HIERARCHY_AUTH_H_UNMARSHAL,
TPM2B_AUTH_P_UNMARSHAL,
END_OF_LIST,
END_OF_LIST}
};
#define _HierarchyChangeAuthDataAddress (&_HierarchyChangeAuthData)
#else
#define _HierarchyChangeAuthDataAddress 0
#endif
// CC_HierarchyChangeAuth
#if CC_DictionaryAttackLockReset
#include "DictionaryAttackLockReset_fp.h"
typedef TPM_RC (DictionaryAttackLockReset_Entry)(
    DictionaryAttackLockReset_In *in
);
typedef const struct {
    DictionaryAttackLockReset_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    BYTE types[3];
} DictionaryAttackLockReset_COMMAND_DESCRIPTOR_t;
DictionaryAttackLockReset_COMMAND_DESCRIPTOR_t _DictionaryAttackLockResetData = {
    /* entry */ &TPM2_DictionaryAttackLockReset,
    /* inSize */ (UINT16)(sizeof(DictionaryAttackLockReset_In)),
    /* outSize */ 0,
    /* offsetOfTypes */
offsetof(DictionaryAttackLockReset_COMMAND_DESCRIPTOR_t, types),
    /* offsets */
{(UINT16)(offsetof(DictionaryAttackLockReset_In, newLockout))},
    /* types */
{TPMI_RH_LOCKOUT_H_UNMARSHAL,
END_OF_LIST,
END_OF_LIST}
};
#define _DictionaryAttackLockResetDataAddress (&_DictionaryAttackLockResetData)
#else
#define _DictionaryAttackLockResetDataAddress 0
#endif
// CC_DictionaryAttackLockReset
# CC_DictionaryAttackLockReset

#define CC_DictionaryAttackLockReset

#endif

#if CC_DictionaryAttackParameters

#include "DictionaryAttackParameters_fp.h"

typedef TPM_RC (DictionaryAttackParameters_Entry)(
    DictionaryAttackParameters_In  *in
);

type const struct {
    DictionaryAttackParameters_Entry  *entry;
    UINT16                              inSize;
    UINT16                              outSize;
    UINT16                              offsetOfTypes;
    UINT16                              paramOffsets[3];
    BYTE                                types[6];
} DictionaryAttackParameters_COMMAND_DESCRIPTOR_t;

DictionaryAttackParameters_COMMAND_DESCRIPTOR_t _DictionaryAttackParametersData = {
    /* entry */ &TPM2_DictionaryAttackParameters,
    /* inSize */ (UINT16)(sizeof(DictionaryAttackParameters_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(DictionaryAttackParameters_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(DictionaryAttackParameters_In, newMaxTries)),
        (UINT16)(offsetof(DictionaryAttackParameters_In, newRecoveryTime))},
    /* types */ {
        TPMI_RH_LOCKOUT_H_UNMARSHAL,
        UINT32_P_UNMARSHAL,
        UINT32_P_UNMARSHAL,
        UINT32_P_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};

#define _DictionaryAttackParametersDataAddress (_DictionaryAttackParametersData)

#else

#define _DictionaryAttackParametersDataAddress 0
#endif

#endif /* CC_DictionaryAttackParameters */

#if CC_PP_Commands

#include "PP_Commands_fp.h"

typedef TPM_RC (PP_Commands_Entry)(
    PP_Commands_In              *in
);

type const struct {
    PP_Commands_Entry       *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[2];
    BYTE                    types[5];
} PP_Commands_COMMAND_DESCRIPTOR_t;

PP_Commands_COMMAND_DESCRIPTOR_t _PP_CommandsData = {
    /* entry */ &TPM2_PP_Commands,
    /* inSize */ (UINT16)(sizeof(PP_Commands_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(PP_Commands_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(PP_Commands_In, setList)),
        (UINT16)(offsetof(PP_Commands_In, clearList))},
    /* types */ {
        TPMI_RH_PLATFORM_H_UNMARSHAL,
        TPML_CC_P_UNMARSHAL,
        TPML_CC_P_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};

#define _PP_CommandsDataAddress (_PP_CommandsData)

#else

#define _PP_CommandsDataAddress (0)
#endif

#endif /* CC_PP_Commands */
#define _PP_CommandsDataAddress 0
#endif // CC_PP_Commands
#if CC_SetAlgorithmSet
#include "SetAlgorithmSet_fp.h"
typedef TPM_RC (SetAlgorithmSet_Entry)(
    SetAlgorithmSet_In *in
);
typedef const struct {
    SetAlgorithmSet_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[4];
} SetAlgorithmSet_COMMAND_DESCRIPTOR_t;
SetAlgorithmSet_COMMAND_DESCRIPTOR_t _SetAlgorithmSetData = {
    /* entry */ &TPM2_SetAlgorithmSet,
    /* inSize */ (UINT16)(sizeof(SetAlgorithmSet_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(SetAlgorithmSet_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {(UINT16)(offsetof(SetAlgorithmSet_In, algorithmSet))},
    /* types */ {TPMI_RH_PLATFORM_H_UNMARSHAL,
                 UINT32_P_UNMARSHAL,
                 END_OF_LIST,
                 END_OF_LIST} }
#define _SetAlgorithmSetDataAddress (&_SetAlgorithmSetData)
#else
#define _SetAlgorithmSetDataAddress 0
#endif // CC_SetAlgorithmSet

#if CC_FieldUpgradeStart
#include "FieldUpgradeStart_fp.h"
typedef TPM_RC (FieldUpgradeStart_Entry)(
    FieldUpgradeStart_In *in
);
typedef const struct {
    FieldUpgradeStart_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[3];
    BYTE types[6];
} FieldUpgradeStart_COMMAND_DESCRIPTOR_t;
FieldUpgradeStart_COMMAND_DESCRIPTOR_t _FieldUpgradeStartData = {
    /* entry */ &TPM2_FieldUpgradeStart,
    /* inSize */ (UINT16)(sizeof(FieldUpgradeStart_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(FieldUpgradeStart_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {(UINT16)(offsetof(FieldUpgradeStart_In, keyHandle)),
                   (UINT16)(offsetof(FieldUpgradeStart_In, fuDigest)),
                   (UINT16)(offsetof(FieldUpgradeStart_In, manifestSignature))},
    /* types */ {TPMI_RH_PLATFORM_H_UNMARSHAL,
                 TPMI_DH_OBJECT_H_UNMARSHAL,
                 TPM2B_DIGEST_P_UNMARSHAL,
                 TPM2_SIGNATURE_P_UNMARSHAL,
                 END_OF_LIST,
                 END_OF_LIST} }
#define _FieldUpgradeStartDataAddress (&_FieldUpgradeStartData)
#else
#define _FieldUpgradeStartDataAddress 0
#endif // CC_FieldUpgradeStart
typedef TPM_RC (FieldUpgradeData_Entry)(
    FieldUpgradeData_In *in,
    FieldUpgradeData_Out *out
);
typedef const struct {
    FieldUpgradeData_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[5];
} FieldUpgradeData_COMMAND_DESCRIPTOR_t;
FieldUpgradeData_COMMAND_DESCRIPTOR_t _FieldUpgradeDataData = {
    /* entry         */ &TPM2_FieldUpgradeData,
    /* inSize        */ (UINT16)(sizeof(FieldUpgradeData_In)),
    /* outSize       */ (UINT16)(sizeof(FieldUpgradeData_Out)),
    /* offsetOfTypes */ offsetOfFieldUpgradeData_COMMAND_DESCRIPTOR_t, types,
    /* offsets */ 
        // No parameter offsets;
    /* types */ 
        // No parameter offsets;
}
#define _FieldUpgradeDataDataAddress (&_FieldUpgradeDataData)
#endif
#endif
// CC_FieldUpgradeData

#if CC_FirmwareRead
#include "FirmwareRead_fp.h"
typedef TPM_RC (FirmwareRead_Entry)(
    FirmwareRead_In *in,
    FirmwareRead_Out *out
);
typedef const struct {
    FirmwareRead_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    BYTE types[4];
} FirmwareRead_COMMAND_DESCRIPTOR_t;
FirmwareRead_COMMAND_DESCRIPTOR_t _FirmwareReadData = {
    /* entry */ &TPM2_FirmwareRead,
    /* inSize */ (UINT16)(sizeof(FirmwareRead_In)),
    /* outSize */ (UINT16)(sizeof(FirmwareRead_Out)),
    /* offsetOfTypes */ offsetOfFirmwareRead_COMMAND_DESCRIPTOR_t, types,
    /* offsets */
        // No parameter offsets;
    /* types */
        // No parameter offsets;
}
#define _FirmwareReadDataAddress (&_FirmwareReadData)
#endif
#endif
#if CC_ContextSave
#include "ContextSave_fp.h"
typedef TPM_RC (ContextSave_Entry)(
    ContextSave_In *in,
    ContextSave_Out *out
);
typedef const struct {
    ContextSave_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
} ContextSave_COMMAND_DESCRIPTOR_t;
ContextSave_COMMAND_DESCRIPTOR_t _ContextSaveData = {
    /* entry */ &TPM2_ContextSave,
    /* inSize */ (UINT16)(sizeof(ContextSave_In)),
    /* outSize */ (UINT16)(sizeof(ContextSave_Out)),
    /* offsetOfTypes */ offsetOfContextSave_COMMAND_DESCRIPTOR_t, types,
    /* offsets */
        // No parameter offsets;
    /* types */
        // No parameter offsets;
}
#define _ContextSaveDataAddress (&_ContextSaveData)
#endif
#else
#define _ContextSaveDataAddress 0
#endif
// CC_ContextSave

```c
3164 #define _ContextSaveDataAddress (&_ContextSaveData)
3165 #else
3166 #define _ContextSaveDataAddress 0
3167 #endif // CC_ContextSave
3168 #if CC_ContextLoad
3169 #include "ContextLoad_fp.h"
3170 typedef TPM_RC (ContextLoad_Entry)(
3171 ContextLoad_In *in,
3172 ContextLoad_Out *out
3173 );
3174 typedef const struct {
3175 ContextLoad_Entry *entry;
3176 UINT16 inSize;
3177 UINT16 outSize;
3178 UINT16 offsetOfTypes;
3179 BYTE types[4];
3180 } ContextLoad_COMMAND_DESCRIPTOR_t;
3181 ContextLoad_COMMAND_DESCRIPTOR_t _ContextLoadData = {
3182 /* entry */ &TPM2_ContextLoad,
3183 /* inSize */ (UINT16)(sizeof(ContextLoad_In)),
3184 /* outSize */ (UINT16)(sizeof(ContextLoad_Out)),
3185 /* offsetOfTypes */ offsetof(ContextLoad_COMMAND_DESCRIPTOR_t, types),
3186 /* offsets */ // No parameter offsets;
3187 /* types */ {TPMS_CONTEXT_P_UNMARSHAL,
3188 TPMI_DH_CONTEXT_H_MARSHAL,
3189 TPMI_DH_CONTEXT_H_UNMARSHAL,
3190 END_OF_LIST,
3191 TPMS_CONTEXT_P_MARSHAL,
3192 END_OF_LIST}
3193};
3194 #define _ContextLoadDataAddress (&_ContextLoadData)
3195 #else
3196 #define _ContextLoadDataAddress 0
3197 #endif // CC_ContextLoad
3198 #if CC_FlushContext
3199 #include "FlushContext_fp.h"
3200 typedef TPM_RC (FlushContext_Entry)(
3201 FlushContext_In *in,
3202 FlushContext_Out *out
3203 );
3204 typedef const struct {
3205 FlushContext_Entry *entry;
3206 UINT16 inSize;
3207 UINT16 outSize;
3208 UINT16 offsetOfTypes;
3209 BYTE types[3];
3210 } FlushContext_COMMAND_DESCRIPTOR_t;
3211 FlushContext_COMMAND_DESCRIPTOR_t _FlushContextData = {
3212 /* entry */ &TPM2_FlushContext,
3213 /* inSize */ (UINT16)(sizeof(FlushContext_In)),
3214 /* outSize */ 0,
3215 /* offsetOfTypes */ offsetof(FlushContext_COMMAND_DESCRIPTOR_t, types),
3216 /* offsets */ // No parameter offsets;
3217 /* types */ {TPMI_DH_CONTEXT_P_UNMARSHAL,
3218 TPMI_DH_CONTEXT_H_MARSHAL,
3219 TPMI_DH_CONTEXT_H_UNMARSHAL,
3220 END_OF_LIST,
3221 TPMI_DH_CONTEXT_P_MARSHAL,
3222 END_OF_LIST}
3223};
3224 #define _FlushContextDataAddress (&_FlushContextData)
3225 #else
3226 #define _FlushContextDataAddress 0
3227 #endif // CC_FlushContext
```
3230inance \_FlushContextDataAddress (\&_FlushContextData)
3231 #else
3232 #define \_FlushContextDataAddress 0
3233 #endif
3234 #include "EvictControl_fp.h"
3235 typedef TPM\_RC (EvictControl\_Entry)(
3236 EvictControl\_In *in
3237);
3238 typedef const struct {
3239 EvictControl\_ENTRY entry;
3240 UINT16 inSize;
3241 UINT16 outSize;
3242 UINT16 offsetOfTypes;
3243 UINT16 paramOffsets[2];
3244 BYTE types[5];
3245 ) EvictControl\_COMMAND\_DESCRIPTOR\_t _EvictControlData = {
3246 /* entry */ \&_TPM2\_EvictControl,
3247 /* inSize */ (UINT16)(sizeof(EvictControl\_In)),
3248 /* outSize */ 0,
3249 /* offsetOfTypes */ offsetOf(EvictControl\_COMMAND\_DESCRIPTOR\_t, types),
3250 /* offsets */ {
3251 (UINT16)(offsetof(EvictControl\_In, objectHandle)),
3252 (UINT16)(offsetof(EvictControl\_In, persistentHandle))},
3253 /* types */ {
3254 TPM\_RH\_PROVISION\_H\_UNMARSHAL,
3255 TPM\_DH\_OBJECT\_H\_UNMARSHAL,
3256 TPM\_DH\_PERSISTENT\_P\_UNMARSHAL,
3257 END\_OF\_LIST,
3258 END\_OF\_LIST
3259};
3260 #define _EvictControlDataAddress (&_EvictControlData)
3261 #else
3262 #define _EvictControlDataAddress 0
3263 #endif
3264 #if CC\_ReadClock
3265 #include "ReadClock_fp.h"
3266 typedef TPM\_RC (ReadClock\_Entry)(
3267 ReadClock\_Out *out
3268);
3269 typedef const struct {
3270 ReadClock\_ENTRY entry;
3271 UINT16 inSize;
3272 UINT16 outSize;
3273 UINT16 offsetOfTypes;
3274 BYTE types[3];
3275 ) ReadClock\_COMMAND\_DESCRIPTOR\_t _ReadClockData = {
3276 /* entry */ \&_TPM2\_ReadClock,
3277 /* inSize */ 0,
3278 /* outSize */ (UINT16)(sizeof(ReadClock\_Out)),
3279 /* offsetOfTypes */ offsetOf(ReadClock\_COMMAND\_DESCRIPTOR\_t, types),
3280 /* offsets */ // No parameter offsets;
3281 /* types */ {
3282 TPM\_RS\_TIME\_INFO\_P\_MARSHAL,
3283 END\_OF\_LIST,
3284 END\_OF\_LIST
3285};
3286 #define _ReadClockDataAddress (&_ReadClockData)
3287 #else
3288 #define _ReadClockDataAddress 0
3289 #endif
3290 #if CC\_ClockSet
3291 #include "ClockSet_fp.h"
3292 typedef TPM\_RC (ClockSet\_Entry)(
3293 ClockSet\_In *in
3294);
typedef const struct {
    ClockSet_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[4];
} ClockSet_COMMAND_DESCRIPTOR_t;

ClockSet_COMMAND_DESCRIPTOR_t _ClockSetData = {
    /* entry         */ &TPM2_ClockSet,
    /* inSize        */ (UINT16)(sizeof(ClockSet_In)),
    /* outSize       */ 0,
    /* offsetOfTypes */ offsetOf(ClockSet_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {(UINT16)(offsetof(ClockSet_In, newTime))},
    /* types         */ {TPMI_RH_PROVISION_H_UNMARSHAL,
                         UINT64_P_UNMARSHAL,
                         END_OF_LIST,
                         END_OF_LIST}
};

#define _ClockSetDataAddress (_ClockSetData)

#else
#define _ClockSetDataAddress 0
#endif // CC_ClockSet

#if CC_ClockRateAdjust
#include "ClockRateAdjust_fp.h"

typedef TPM_RC (ClockRateAdjust_Entry)(
    ClockRateAdjust_In *in
);

typedef const struct {
    ClockRateAdjust_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[4];
} ClockRateAdjust_COMMAND_DESCRIPTOR_t;

ClockRateAdjust_COMMAND_DESCRIPTOR_t _ClockRateAdjustData = {
    /* entry         */ &TPM2_ClockRateAdjust,
    /* inSize        */ (UINT16)(sizeof(ClockRateAdjust_In)),
    /* outSize       */ 0,
    /* offsetOfTypes */ offsetOf(ClockRateAdjust_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {(UINT16)(offsetof(ClockRateAdjust_In, rateAdjust))},
    /* types         */ {TPMI_RH_PROVISION_H_UNMARSHAL,
                         TPM_CLOCK_ADJUST_P_UNMARSHAL,
                         END_OF_LIST,
                         END_OF_LIST}
};

#define _ClockRateAdjustDataAddress (_ClockRateAdjustData)

#else
#define _ClockRateAdjustDataAddress 0
#endif // CC_ClockRateAdjust

#if CC_GetCapability
#include "GetCapability_fp.h"

typedef TPM_RC (GetCapability_Entry)(
    GetCapability_In *in,
    GetCapability_Out *out
);

typedef const struct {
    GetCapability_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[3];
    BYTE types[7];
} GetCapability_COMMAND_DESCRIPTOR_t;
Trusted Platform Module Library

Part 4: Supporting Routines

GetCapability_COMMAND_DESCRIPTOR_t _GetCapabilityData = {
    /* entry */ &TPM2_GetCapability,
    /* inSize */ (UINT16)(sizeof(GetCapability_In)),
    /* outSize */ (UINT16)(sizeof(GetCapability_Out)),
    /* offsetOfTypes */ offset(GetCapability_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(GetCapability_In, property)),
        (UINT16)(offsetof(GetCapability_In, propertyCount)),
        (UINT16)(offsetof(GetCapability_Out, capabilityData))},
    /* types */ {TPM_CAP_P_UNMARSHAL,
        TPM32_P_UNMARSHAL,
        TPM32_P_UNMARSHAL,
        TPM32_NO_P_MARSHAL,
        TPMS_CAPABILITY_DATA_P_MARSHAL,
        END_OF_LIST}
};
#define _GetCapabilityDataAddress (&_GetCapabilityData)
#else
#define _GetCapabilityDataAddress 0
#endif // CC_GetCapability
#endif // CC_TestParms
#if CC_NV_DefineSpace
#include "NV_DefineSpace_fp.h"
typedef TPM_RC (NV_DefineSpace_Entry)(
    NV_DefineSpace_In *in
);
typedef const struct {
    NV_DefineSpace_Entry     *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    BYTE                    types[5];
} NV_DefineSpace_COMMAND_DESCRIPTOR_t;
NV_DefineSpace_COMMAND_DESCRIPTOR_t _NV_DefineSpaceData = {
    /* entry */ &TPM2_NV_DefineSpace,
    /* inSize */ (UINT16)(sizeof(NV_DefineSpace_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offset(NV_DefineSpace_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(NV_DefineSpace_In, auth)),
        (UINT16)(offsetof(NV_DefineSpace_In, publicInfo))},
};
#define _NV_DefineSpaceDataAddress (&_NV_DefineSpaceData)
#else
#define _NV_DefineSpaceDataAddress 0
#endif // CC_NV_DefineSpace

#include "TestParms_fp.h"
typedef struct {
    TestParms_Entry         *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    BYTE                    types[3];
} TestParms_COMMAND_DESCRIPTOR_t;
TestParms_COMMAND_DESCRIPTOR_t _TestParmsData = {
    /* entry */ &TPM2_TestParms,
    /* inSize */ (UINT16)(sizeof(TestParms_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offset(TestParms_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(TestParms_In, property)),
        (UINT16)(offsetof(TestParms_In, propertyCount))},
    /* types */ {TPMT_PUBLIC_PARMS_P_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST}
};
#define _TestParmsDataAddress (&_TestParmsData)
#else
#define _TestParmsDataAddress 0
#endif // CC_TestParms

Page 76  TCG Public Review  Family “2.0”
September 18, 2018  Copyright © TCG 2006-2019  Level 00 Revision 01.50
/* types */
{TPMI_RH_PROVISION_H_UNMARSHAL,
 TPM2B_AUTH_P_UNMARSHAL,
 TPM2B_NV_PUBLIC_P_UNMARSHAL,
 END_OF_LIST,
 END_OF_LIST}
);
#define _NV_DefineSpaceDataAddress (&_NV_DefineSpaceData)
#else
#define _NV_DefineSpaceDataAddress 0
#endif // CC_NV_DefineSpace
#if CC_NV_UndefineSpace
#include "NV_UndefineSpace_fp.h"
typedef TPM_RC (NV_UndefineSpace_Entry)(
 NV_UndefineSpace_In *in
)
 NV_UndefineSpaceCOMMAND_DESCRIPTOR_t _NV_UndefineSpaceData = {
 /* entry */ &TPM2_NV_UndefineSpace,
 /* inSize */ (UINT16)(sizeof(NV_UndefineSpace_In)),
 /* outSize */ 0,
 /* offsetOfTypes */ offsetOf(NV_UndefineSpace_COMMAND_DESCRIPTOR_t, types),
 /* offsets */ 
{((UINT16)(offsetof(NV_UndefineSpace_In, nvIndex)))},
 /* types */ 
{TPMI_RH_PROVISION_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 END_OF_LIST,
 END_OF_LIST}
};
#define _NV_UndefineSpaceDataAddress (&_NV_UndefineSpaceData)
#else
#define _NV_UndefineSpaceDataAddress 0
#endif // CC_NV_UndefineSpace
#if CC_NV_UndefineSpaceSpecial
#include "NV_UndefineSpaceSpecial_fp.h"
typedef TPM_RC (NV_UndefineSpaceSpecial_Entry)(
 NV_UndefineSpaceSpecial_In *in
)
 NV_UndefineSpaceSpecialCOMMAND_DESCRIPTOR_t _NV_UndefineSpaceSpecialData = {
 /* entry */ &TPM2_NV_UndefineSpaceSpecial,
 /* inSize */ (UINT16)(sizeof(NV_UndefineSpaceSpecial_In)),
 /* outSize */ 0,
 /* offsetOfTypes */ offsetOf(NV_UndefineSpaceSpecial_COMMAND_DESCRIPTOR_t, types),
 /* offsets */ 
{((UINT16)(offsetof(NV_UndefineSpaceSpecial_In, platform)))},
 /* types */ 
{TPMI_RH_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_NV_INDEX_H_UNMARSHAL,
 TPM2BPLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
 TPM2B_PLATFORM_H_UNMARSHAL,
#define _NV.UndefineSpaceSpecialDataAddress 0
#endif
// CC_NV.UndefineSpaceSpecial

typedef TPM_RC (NV_ReadPublic_Entry)(
    NV_ReadPublic_In   *in,
    NV_ReadPublic_Out  *out
);

typedef const struct {
    NV_ReadPublic_Entry     *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[1];
    BYTE                    types[5];
    } NV_ReadPublic_COMMAND_DESCRIPTOR_t;

NV_ReadPublic_COMMAND_DESCRIPTOR_t _NV_ReadPublicData = {
    /* entry         */ &TPM2_NV_ReadPublic,
    /* inSize        */ (UINT16)(sizeof(NV_ReadPublic_In)),
    /* offsetOfTypes */ offsetof(NV_ReadPublic_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {
        (UINT16)(offsetof(NV_ReadPublic_Out, nvName)),
        (UINT16)(offsetof(NV_ReadPublic_Out, nvIndex))},
    /* types         */ {
        TPM1_RH_NV_INDEX_H_UNMARSHAL,
        TPM2B_NV_INDEX_H_MARSHAL,
        TPM2B_NV_READ_PUBLIC_P_MARSHAL,
        TPM2B_NV_PUBLIC_P_MARSHAL,
        TPM2B_NAME_P_MARSHAL,
        END_OF_LIST}
};

#define _NV_ReadPublicDataAddress (&_NV_ReadPublicData)
#else
#define _NV_ReadPublicDataAddress 0
#endif
// CC_NV.ReadPublic

#include "NV_Write_fp.h"

typedef TPM_RC (NV_Write_Entry)(
    NV_Write_In   *in
);

typedef const struct {
    NV_Write_Entry          *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[3];
    BYTE                    types[6];
    } NV_Write_COMMAND_DESCRIPTOR_t;

NV_Write_COMMAND_DESCRIPTOR_t _NV_WriteData = {
    /* entry         */ &TPM2_NV_Write,
    /* inSize        */ (UINT16)(sizeof(NV_Write_In)),
    /* outSize       */ 0,
    /* offsetOfTypes */ offsetof(NV_Write_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {
        (UINT16)(offsetof(NV_Write_In, nvIndex)),
        (UINT16)(offsetof(NV_Write_In, data)),
        (UINT16)(offsetof(NV_Write_In, offset))},
    /* types         */ {
        TPM1_RH_NV_AUTH_H_UNMARSHAL,
        TPM1_RH_NV_INDEX_H_UNMARSHAL,
        TPM2B_MAX_NV_BUFFER_P_UNMARSHAL,
        TPM2B_NV_PUBLIC_P_MARSHAL,
        TPM2B_NV_INDEX_H_MARSHAL,
        END_OF_LIST}
};

#define _NV_WriteDataAddress (&_NV_WriteData)
#else
#define _NV_WriteDataAddress 0
#endif
// CC_NV.Write

#include "NV_Increment_fp.h"
typedef TPM_RC (NV_Increment_Entry)(
    NV_Increment_In             *in
);

typedef const struct {
    NV_Increment_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[1];
    BYTE                        types[4];
} NV_Increment_COMMAND_DESCRIPTOR_t;

NV_Increment_COMMAND_DESCRIPTOR_t _NV_IncrementData = {
    /* entry         */ &TPM2_NV_Increment,
    /* inSize        */ (UINT16)(sizeof(NV_Increment_In)),
    /* outSize       */ 0,
    /* offsetOfTypes */ offsetof(NV_Increment_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {(UINT16)(offsetof(NV_Increment_In, nvIndex))},
    /* types         */ {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                             TPM1_RH_NV_INDEX_H_UNMARSHAL,
                             END_OF_LIST,
                             END_OF_LIST}
};
#define _NV_IncrementDataAddress (&_NV_IncrementData)

#else
#define _NV_IncrementDataAddress 0
#endif

#if CC_NV_Extend
#include "NV_Extend_fp.h"
typedef TPM_RC (NV_Extend_Entry)(
    NV_Extend_In             *in
);

typedef const struct {
    NV_Extend_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
    UINT16                      paramOffsets[2];
    BYTE                        types[5];
} NV_Extend_COMMAND_DESCRIPTOR_t;

NV_Extend_COMMAND_DESCRIPTOR_t _NV_ExtendData = {
    /* entry         */ &TPM2_NV_Extend,
    /* inSize        */ (UINT16)(sizeof(NV_Extend_In)),
    /* outSize       */ 0,
    /* offsetOfTypes */ offsetof(NV_Extend_COMMAND_DESCRIPTOR_t, types),
    /* offsets       */ {(UINT16)(offsetof(NV_Extend_In, nvIndex))},
    /* types         */ {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                             TPM1_RH_NV_INDEX_H_UNMARSHAL,
                             TPM2B_MAX_NV_BUFFER_P_UNMARSHAL,
                             END_OF_LIST,
                             END_OF_LIST}
};
#define _NV_ExtendDataAddress (&_NV_ExtendData)

#else
#define _NV_ExtendDataAddress 0
#endif

#if CC_NV_SetBits
#include "NV_SetBits_fp.h"
typedef TPM_RC (NV_SetBits_Entry)(
    NV_SetBits_In             *in
);

typedef const struct {
    NV_SetBits_Entry         *entry;
    UINT16                      inSize;
    UINT16                      outSize;
    UINT16                      offsetOfTypes;
3624 UINT16 paramOffsets[2];
3625 BYTE types[5];
3626 } NV_SetBits_COMMAND_DESCRIPTOR_t;
3627 NV_SetBits_COMMAND_DESCRIPTOR_t _NV_SetBitsData = {
3628    /* entry */ &TPM2_NV_SetBits,
3629    /* inSize */ (UINT16)(sizeof(NV_SetBits_In)),
3630    /* outSize */ 0,
3631    /* offsetOfTypes */ offsetof(NV_SetBits_COMMAND_DESCRIPTOR_t, types),
3632    /* offsets */ {(UINT16)(offsetof(NV_SetBits_In, nvIndex)),
3633    /* types */ {TPMI_RH_NV_AUTH_H_UNMARSHAL,
3634        TPMI_RH_NV_INDEX_H_UNMARSHAL,
3635        UINT64_P_UNMARSHAL,
3636        END_OF_LIST,
3637        END_OF_LIST}
3638};
3639 #define _NV_SetBitsDataAddress (&_NV_SetBitsData)
3640 #else
3641 #define _NV_SetBitsDataAddress 0
3642 #endif // CC_NV_SetBits
3643 #if CC_NV_WriteLock
3644 #include "NV_WriteLock_fp.h"
3645 typedef TPM_RC (NV_WriteLock_Entry)(
3646    NV_WriteLock_In *in
3647 );
3648 typedef const struct {
3649    NV_WriteLock_Entry *entry;
3650    UINT16 inSize;
3651    UINT16 outSize;
3652    UINT16 offsetOfTypes;
3653    UINT16 paramOffsets[1];
3654    BYTE types[4];
3655 } NV_WriteLock_COMMAND_DESCRIPTOR_t;
3656 NV_WriteLock_COMMAND_DESCRIPTOR_t _NV_WriteLockData = {
3657    /* entry */ &TPM2_NV_WriteLock,
3658    /* inSize */ (UINT16)(sizeof(NV_WriteLock_In)),
3659    /* outSize */ 0,
3660    /* offsetOfTypes */ offsetof(NV_WriteLock_COMMAND_DESCRIPTOR_t, types),
3661    /* offsets */ {(UINT16)(offsetof(NV_WriteLock_In, nvIndex))},
3662    /* types */ {TPMI_RH_NV_AUTH_H_UNMARSHAL,
3663        TPMI_RH_NV_INDEX_H_UNMARSHAL,
3664        END_OF_LIST,
3665        END_OF_LIST}
3666};
3667 #define _NV_WriteLockDataAddress (&_NV_WriteLockData)
3668 #else
3669 #define _NV_WriteLockDataAddress 0
3670 #endif // CC_NV_WriteLock
3671 #if CC_NV_GlobalWriteLock
3672 #include "NV_GlobalWriteLock_fp.h"
3673 typedef TPM_RC (NV_GlobalWriteLock_Entry)(
3674    NV_GlobalWriteLock_In *in
3675 );
3676 typedef const struct {
3677    NV_GlobalWriteLock_Entry    *entry;
3678    UINT16                      inSize;
3679    UINT16                      outSize;
3680    UINT16                      offsetOfTypes;
3681    BYTE                        types[3];
3682 } NV_GlobalWriteLock_COMMAND_DESCRIPTOR_t;
3683 NV_GlobalWriteLock_COMMAND_DESCRIPTOR_t _NV_GlobalWriteLockData = {
3684    /* entry */ &TPM2_NV_GlobalWriteLock,
3685    /* inSize */ (UINT16)(sizeof(NV_GlobalWriteLock_In)),
3686    /* outSize */ 0,
3687    /* offsetOfTypes */ offsetof(NV_GlobalWriteLock_COMMAND_DESCRIPTOR_t, types),
/* offsets */   // No parameter offsets:
/* types */    {TPMI_RH_PROVISION_H_UNMARSHAL,
                        END_OF_LIST,
                        END_OF_LIST}
};
define _NV_GlobalWriteLockDataAddress (&_NV_GlobalWriteLockData)
define _NV_GlobalWriteLockDataAddress 0
endif // CC_NV_GlobalWriteLock

if CC_NV_Read
#include "NV_Read_fp.h"
typedef TPM_RC (NV_Read_Entry) ( NV_Read_In *in,
                          NV_Read_Out *out
                        )
#endif

if CC_NV_ReadLock
#define _NV_ReadDataAddress (&_NV_ReadData)
define _NV_ReadDataAddress _NV_ReadData = {
                        /* entry */    &TPM2_NV_Read,
                        /* inSize */   (UINT16)(sizeof(NV_Read_In)),
                        /* outSize */  (UINT16)(sizeof(NV_Read_Out)),
                        /* offsetOfTypes */   offsetof(NV_Read_COMMAND_DESCRIPTOR_t, types),
                        /* offsets */    {(UINT16)(offsetof(NV_Read_In, nvIndex))},
                        /* types */     {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                          TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                          UINT16_P_UNMARSHAL,
                                          UINT16_P_UNMARSHAL,
                                          TPM2B_MAX_NV_BUFFER_P_MARSHAL,
                                          END_OF_LIST}
                        }
#endif

if CC_NV_ReadLock
#include "NV_ReadLock_fp.h"
typedef TPM_RC (NV_ReadLock_Entry) ( NV_ReadLock_In *in
                                )
#endif

if CC_NV_ReadLock
#define _NV_ReadLockDataAddress (&_NV_ReadLockData)
define _NV_ReadLockDataAddress 0
endif // CC_NV_Read

if CC_NV_ReadLock
#include "NV_ReadLock_fp.h"
typedef TPM_RC (NV_ReadLock_Entry) ( NV_ReadLock_In *in
                                )
#endif

typedef const struct {
                        NV_Read_Entry *entry;
                        UINT16 inSize;
                        UINT16 outSize;
                        UINT16 offsetOfTypes;
                        UINT16 paramOffsets[3];
                        BYTE types[7];
                        } NV_Read_COMMAND_DESCRIPTOR_t;
NV_Read_COMMAND_DESCRIPTOR_t _NV_ReadData = {
                        /* entry */    &TPM2_NV_Read,
                        /* inSize */   (UINT16)(sizeof(NV_Read_In)),
                        /* outSize */  0,
                        /* offsetOfTypes */   offsetof(NV_Read_COMMAND_DESCRIPTOR_t, types),
                        /* offsets */    {(UINT16)(offsetof(NV_Read_In, nvIndex))},
                        /* types */     {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                          TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                          END_OF_LIST}
                        }
#endif

if CC_NV_ReadLock
#define _NV_ReadLockDataAddress (&_NV_ReadLockData)
define _NV_ReadLockDataAddress 0
endif // CC_NV_Read

if CC_NV_ReadLock
#include "NV_ReadLock_fp.h"
typedef TPM_RC (NV_ReadLock_Entry) ( NV_ReadLock_In *in
                                )
#endif

typedef const struct {
                        NV_ReadLock_Entry *entry;
                        UINT16 inSize;
                        UINT16 outSize;
                        UINT16 offsetOfTypes;
                        UINT16 paramOffsets[1];
                        BYTE types[4];
                        } NV_ReadLock_COMMAND_DESCRIPTOR_t;
NV_ReadLock_COMMAND_DESCRIPTOR_t _NV_ReadLockData = {
                        /* entry */    &TPM2_NV_ReadLock,
                        /* inSize */   (UINT16)(sizeof(NV_ReadLock_In)),
                        /* outSize */  0,
                        /* offsetOfTypes */   offsetof(NV_ReadLock_COMMAND_DESCRIPTOR_t, types),
                        /* offsets */    {(UINT16)(offsetof(NV_ReadLock_In, nvIndex))},
                        /* types */     {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                          TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                          END_OF_LIST}
                        }
#endif

typedef const struct {
                        NV_ReadEntry *entry;
                        UINT16 inSize;
                        UINT16 outSize;
                        UINT16 offsetOfTypes;
                        UINT16 paramOffsets[3];
                        BYTE types[7];
                        } NV_ReadCommandDescriptor_t;
NV_ReadCommandDescriptor_t _NV_ReadData = {
                        /* entry */    &TPM2_NV_Read,
                        /* inSize */   (UINT16)(sizeof(NV_Read_In)),
                        /* outSize */  (UINT16)(sizeof(NV_Read_Out)),
                        /* offsetOfTypes */   offsetof(NV_Read_COMMAND_DESCRIPTOR_t, types),
                        /* offsets */    {(UINT16)(offsetof(NV_Read_In, nvIndex))},
                        /* types */     {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                          TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                          END_OF_LIST}
                        }
#endif

typedef const struct {
                        NV_ReadLockEntry *entry;
                        UINT16 inSize;
                        UINT16 outSize;
                        UINT16 offsetOfTypes;
                        UINT16 paramOffsets[1];
                        BYTE types[4];
                        } NV_ReadLockCommandDescriptor_t;
NV_ReadLockCommandDescriptor_t _NV_ReadLockData = {
                        /* entry */    &TPM2_NV_ReadLock,
                        /* inSize */   (UINT16)(sizeof(NV_ReadLock_In)),
                        /* outSize */  0,
                        /* offsetOfTypes */   offsetof(NV_ReadLock_COMMAND_DESCRIPTOR_t, types),
                        /* offsets */    {(UINT16)(offsetof(NV_ReadLock_In, nvIndex))},
                        /* types */     {TPMI_RH_NV_AUTH_H_UNMARSHAL,
                                          TPMI_RH_NV_INDEX_H_UNMARSHAL,
                                          END_OF_LIST}
                        }
#endif
```c
#define _NV_ReadLockDataAddress (&_NV_ReadLockData)
#else
#define _NV_ReadLockDataAddress 0
#endif // CC_NV_ReadLock

#include "NV_ChangeAuth_fp.h"

typedef TPM_RC (NV_ChangeAuth_Entry)(
    NV_ChangeAuth_In *in
    NV_ChangeAuth_In
)*in;

typedef const struct {
    NV_ChangeAuth_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[1];
    BYTE types[4];
} NV_ChangeAuth_COMMAND_DESCRIPTOR_t;

NV_ChangeAuth_COMMAND_DESCRIPTOR_t _NV_ChangeAuthData = {
    /* entry */ &TPM2_NV_ChangeAuth,
    /* inSize */ (UINT16)(sizeof(NV_ChangeAuth_In)),
    /* outSize */ 0,
    /* offsetOfTypes */ offsetof(NV_ChangeAuth_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(NV_ChangeAuth_In, newAuth)),
    }
    /* types */ {
        TPM2B_AUTH_P_UNMARSHAL,
        TPM2B_DATA_P_UNMARSHAL,
        END_OF_LIST,
        END_OF_LIST
    }
};
#define _NV_ChangeAuthDataAddress (&_NV_ChangeAuthData)
#else
#define _NV_ChangeAuthDataAddress 0
#endif // CC_NV_ChangeAuth

#include "NV_Certify_fp.h"

typedef TPM_RC (NV_Certify_Entry)(
    NV_Certify_In               *in,
    NV_Certify_Out              *out
    NV_Certify_In
)*in,
    NV_Certify_Out
)*out;

typedef const struct {
    NV_Certify_Entry        *entry;
    UINT16                  inSize;
    UINT16                  outSize;
    UINT16                  offsetOfTypes;
    UINT16                  paramOffsets[7];
    BYTE                    types[11];
} NV_Certify_COMMAND_DESCRIPTOR_t;

NV_Certify_COMMAND_DESCRIPTOR_t _NV_CertifyData = {
    /* entry */ &TPM2_NV_Certify,
    /* inSize */ (UINT16)(sizeof(NV_Certify_In)),
    /* outSize */ (UINT16)(sizeof(NV_Certify_Out)),
    /* offsetOfTypes */ offsetof(NV_Certify_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(NV_Certify_In, authHandle)),
        (UINT16)(offsetof(NV_Certify_In, nvIndex)),
        (UINT16)(offsetof(NV_Certify_In, qualifyingData)),
        (UINT16)(offsetof(NV_Certify_In, inScheme)),
        (UINT16)(offsetof(NV_Certify_In, size)),
        (UINT16)(offsetof(NV_Certify_In, offset)),
        (UINT16)(offsetof(NV_Certify_Out, signature)),
    }
    /* types */ {
        TPM2B_DH_OBJECT_H_UNMARSHAL + ADD_FLAG,
        TPM2B_NV_AUTH_H_UNMARSHAL,
        TPM2B_NV_INDEX_H_UNMARSHAL,
        TPM2B_DATA_P_UNMARSHAL,
        TPM2B_SIG_SCHEME_P_UNMARSHAL + ADD_FLAG,
        UINT16_P_UNMARSHAL,
        UINT16_P_UNMARSHAL,
};
```
typedef TPM_RC (AC_GetCapability_Entry) (AC_GetCapability_In *in, AC_GetCapability_Out *out);

typedef const struct {
    AC_GetCapability_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[3];
    BYTE types[7];
} AC_GetCapability_COMMAND_DESCRIPTOR_t;

AC_GetCapability_COMMAND_DESCRIPTOR_t _AC_GetCapabilityData = {
    /* entry */ &TPM2_AC_GetCapability,
    /* inSize */ (UINT16)(sizeof(AC_GetCapability_In)),
    /* outSize */ (UINT16)(sizeof(AC_GetCapability_Out)),
    /* offsetOfTypes */ offsetOf(AC_GetCapability_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(AC_GetCapability_In, capability)),
        (UINT16)(offsetof(AC_GetCapability_In, count)),
        (UINT16)(offsetof(AC_GetCapability_Out, capabilitiesData))},
    /* types */ {
        TPMI_RH_AC_H_UNMASHAL,
        TPM_AT_P_UNMASHAL,
        UINT32_P_UNMASHAL,
        END_OF_LIST,
        TPMI_YES_NO_P_MARSHAL,
        TPM_AC_CAPABILITIES_P_MARSHAL,
        END_OF_LIST}
};

#define _AC_GetCapabilityDataAddress (&_AC_GetCapabilityData)

if CC_AC_Send
#include "AC_Send_fp.h"
typedef TPM_RC (AC_Send_Entry) (AC_Send_In *in, AC_Send_Out *out);

typedef const struct {
    AC_Send_Entry *entry;
    UINT16 inSize;
    UINT16 outSize;
    UINT16 offsetOfTypes;
    UINT16 paramOffsets[3];
    BYTE types[7];
} AC_Send_COMMAND_DESCRIPTOR_t;

AC_Send_COMMAND_DESCRIPTOR_t _AC_SendData = {
    /* entry */ &TPM2_AC_Send,
    /* inSize */ (UINT16)(sizeof(AC_Send_In)),
    /* outSize */ (UINT16)(sizeof(AC_Send_Out)),
    /* offsetOfTypes */ offsetOf(AC_Send_COMMAND_DESCRIPTOR_t, types),
    /* offsets */ {
        (UINT16)(offsetof(AC_Send_In, authHandle)),
        (UINT16)(offsetof(AC_Send_In, ac)),
        (UINT16)(offsetof(AC_Send_Out, acDataIn))},
};

#define _AC_SendDataAddress (&_AC_SendData)

else
#define _AC_SendDataAddress (&_AC_SendData)
#define _Policy_AC_SendSelectDataAddress (&_Policy_AC_SendSelectData)
#define _Vendor_TCG_TestDataAddress (&_Vendor_TCG_TestData)
/* outSize */ (UINT16)(sizeof(Vendor_TCG_Test_Out)),
/* offsetOfTypes */ offsetof(Vendor_TCG_Test_COMMAND_DESCRIPTOR_t, types),
/* offsets */ // No parameter offsets;
/* types */ {TPM2B_DATA_P_UNMARSHAL, 
END_OF_LIST, 
TPM2B_DATA_P_MARSHAL, 
END_OF_LIST} 
;
#define _Vendor_TCG_TestDataAddress (&_Vendor_TCG_TestData)
#else
#define _Vendor_TCG_TestDataAddress 0
#endif // CC_Vendor_TCG_Test
COMMAND_DESCRIPTOR_t *s_CommandDataArray[] = {
#if (PAD_LIST || CC_NV_UndefineSpaceSpecial)
(COMMAND_DESCRIPTOR_t *)_NV_UndefineSpaceSpecialDataAddress, 
#endif // CC_NV_UndefineSpaceSpecial 
#if (PAD_LIST || CC_EvictControl)
(COMMAND_DESCRIPTOR_t *)_EvictControlDataAddress, 
#endif // CC_EvictControl 
#if (PAD_LIST || CC_HierarchyControl)
(COMMAND_DESCRIPTOR_t *)_HierarchyControlDataAddress, 
#endif // CC_HierarchyControl 
#if (PAD_LIST || CC_NV_UndefineSpace)
(COMMAND_DESCRIPTOR_t *)_NV_UndefineSpaceDataAddress, 
#endif // CC_NV_UndefineSpace 
#if (PAD_LIST)
(COMMAND_DESCRIPTOR_t *)0, 
#endif // CC_PCR_Allocate 
#if (PAD_LIST || CC_PCR_SetAuthPolicy)
(COMMAND_DESCRIPTOR_t *)_PCR_SetAuthPolicyDataAddress, 
#endif // CC_PCR_SetAuthPolicy 
#if (PAD_LIST || CC_PP_Commands)
(COMMAND_DESCRIPTOR_t *)_PP_CommandsDataAddress, 
#endif // CC_PP_Commands 
#if (PAD_LIST || CC_SetPrimaryPolicy)
(COMMAND_DESCRIPTOR_t *)_SetPrimaryPolicyDataAddress, 
#endif // CC_SetPrimaryPolicy 
#if (PAD_LIST || CC_FieldUpgradeStart)
(COMMAND_DESCRIPTOR_t *)_FieldUpgradeStartDataAddress, 
#endif // CC_FieldUpgradeStart 
#if (PAD_LIST || CC_ClockRateAdjust)
(COMMAND_DESCRIPTOR_t *)_ClockRateAdjustDataAddress,
4014 #endif // CC_ClockRateAdjust
4015 #if (PAD_LIST || CC_CreatePrimary)
4016 (COMMAND_DESCRIPTOR_t *)_CreatePrimaryDataAddress,
4017 #endif // CC_CreatePrimary
4018 #if (PAD_LIST || CC_NV_GlobalWriteLock)
4019 (COMMAND_DESCRIPTOR_t *)_NV_GlobalWriteLockDataAddress,
4020 #endif // CC_NV_GlobalWriteLock
4021 #if (PAD_LIST || CC_GetCommandAuditDigest)
4022 (COMMAND_DESCRIPTOR_t *)_GetCommandAuditDigestDataAddress,
4023 #endif // CC_GetCommandAuditDigest
4024 #if (PAD_LIST || CC_NV_Increment)
4025 (COMMAND_DESCRIPTOR_t *)_NV_IncrementDataAddress,
4026 #endif // CC_NV_Increment
4027 #if (PAD_LIST || CC_NV_SetBits)
4028 (COMMAND_DESCRIPTOR_t *)_NV_SetBitsDataAddress,
4029 #endif // CC_NV_SetBits
4030 #if (PAD_LIST || CC_NV_Extend)
4031 (COMMAND_DESCRIPTOR_t *)_NV_ExtendDataAddress,
4032 #endif // CC_NV_Extend
4033 #if (PAD_LIST || CC_NV_WriteLock)
4034 (COMMAND_DESCRIPTOR_t *)_NV_WriteLockDataAddress,
4035 #endif // CC_NV_WriteLock
4036 #if (PAD_LIST || CC_NV_Write)
4037 (COMMAND_DESCRIPTOR_t *)_NV_WriteDataAddress,
4038 #endif // CC_NV_Write
4039 #if (PAD_LIST || CC_DictionaryAttackLockReset)
4040 (COMMAND_DESCRIPTOR_t *)_DictionaryAttackLockResetDataAddress,
4041 #endif // CC_DictionaryAttackLockReset
4042 #if (PAD_LIST || CC_DictionaryAttackParameters)
4043 (COMMAND_DESCRIPTOR_t *)_DictionaryAttackParametersDataAddress,
4044 #endif // CC_DictionaryAttackParameters
4045 #if (PAD_LIST || CC_NV_ChangeAuth)
4046 (COMMAND_DESCRIPTOR_t *)_NV_ChangeAuthDataAddress,
4047 #endif // CC_NV_ChangeAuth
4048 #if (PAD_LIST || CC_PCR_Event)
4049 (COMMAND_DESCRIPTOR_t *)_PCR_EventDataAddress,
4050 #endif // CC_PCR_Event
4051 #if (PAD_LIST || CC_PCR_Reset)
4052 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4053 #endif // CC_PCR_Reset
4054 #if (PAD_LIST || CC_SequenceComplete)
4055 (COMMAND_DESCRIPTOR_t *)_SequenceCompleteDataAddress,
4056 #endif // CC_SequenceComplete
4057 #if (PAD_LIST || CC_PCR_Reset)
4058 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4059 #endif // CC_PCR_Reset
4060 #if (PAD_LIST || CC_PCR_Reset)
4061 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4062 #endif // CC_PCR_Reset
4063 #if (PAD_LIST || CC_PCR_Reset)
4064 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4065 #endif // CC_PCR_Reset
4066 #if (PAD_LIST || CC_PCR_Reset)
4067 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4068 #endif // CC_PCR_Reset
4069 #if (PAD_LIST || CC_PCR_Reset)
4070 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4071 #endif // CC_PCR_Reset
4072 #if (PAD_LIST || CC_PCR_Reset)
4073 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4074 #endif // CC_PCR_Reset
4075 #if (PAD_LIST || CC_PCR_Reset)
4076 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4077 #endif // CC_PCR_Reset
4078 #if (PAD_LIST || CC_PCR_Reset)
4079 (COMMAND_DESCRIPTOR_t *)_PCR_ResetDataAddress,
4080 #endif // CC_StirRandom
4081 #if (PAD_LIST || CC_ActivateCredential)
4082 (COMMAND_DESCRIPTOR_t *)_ActivateCredentialDataAddress,
4083 #endif // CC_ActivateCredential
4084 #if (PAD_LIST || CC_Certify)
4085 (COMMAND_DESCRIPTOR_t *)_CertifyDataAddress,
4086 #endif // CC_Certify
4087 #if (PAD_LIST || CC_PolicyNV)
4088 (COMMAND_DESCRIPTOR_t *)_PolicyNVDataAddress,
4089 #endif // CC_PolicyNV
4090 #if (PAD_LIST || CC_CertifyCreation)
4091 (COMMAND_DESCRIPTOR_t *)_CertifyCreationDataAddress,
4092 #endif // CC_CertifyCreation
4093 #if (PAD_LIST || CC_Duplicate)
4094 (COMMAND_DESCRIPTOR_t *)_DuplicateDataAddress,
4095 #endif // CC_Duplicate
4096 #if (PAD_LIST || CC_GetTime)
4097 (COMMAND_DESCRIPTOR_t *)_GetTimeDataAddress,
4098 #endif // CC_GetTime
4099 #if (PAD_LIST || CC_GetSessionAuditDigest)
4100 (COMMAND_DESCRIPTOR_t *)_GetSessionAuditDigestDataAddress,
4101 #endif // CC_GetSessionAuditDigest
4102 #if (PAD_LIST || CC_NV_Read)
4103 (COMMAND_DESCRIPTOR_t *)_NV_ReadDataAddress,
4104 #endif // CC_NV_Read
4105 #if (PAD_LIST || CC_NV_ReadLock)
4106 (COMMAND_DESCRIPTOR_t *)_NV_ReadLockDataAddress,
4107 #endif // CC_NV_ReadLock
4108 #if (PAD_LIST || CC_ObjectChangeAuth)
4109 (COMMAND_DESCRIPTOR_t *)_ObjectChangeAuthDataAddress,
4110 #endif // CC_ObjectChangeAuth
4111 #if (PAD_LIST || CC_PolicySecret)
4112 (COMMAND_DESCRIPTOR_t *)_PolicySecretDataAddress,
4113 #endif // CC_PolicySecret
4114 #if (PAD_LIST || CC_Rewrap)
4115 (COMMAND_DESCRIPTOR_t *)_RewrapDataAddress,
4116 #endif // CC_Rewrap
4117 #if (PAD_LIST || CC_Create)
4118 (COMMAND_DESCRIPTOR_t *)_CreateDataAddress,
4119 #endif // CC_Create
4120 #if (PAD_LIST || CC_ECDH_ZGen)
4121 (COMMAND_DESCRIPTOR_t *)_ECDH_ZGenDataAddress,
4122 #endif // CC_ECDH_ZGen
4123 #if (PAD_LIST || (CC_HMAC || CC_MAC))
4124 # if CC_HMAC
4125 (COMMAND_DESCRIPTOR_t *)_HMACDataAddress,
4126 # endif
4127 # if CC_MAC
4128 (COMMAND_DESCRIPTOR_t *)_MACDataAddress,
4129 # endif
4130 # if (CC_HMAC || CC_MAC) > 1
4131 # error "More than one aliased command defined"
4132 # endif
4133 #endif // CC_HMAC CC_MAC
4134 #if (PAD_LIST || CC_Import)
4135 (COMMAND_DESCRIPTOR_t *)_ImportDataAddress,
4136 #endif // CC_Import
4137 #if (PAD_LIST || CC_Load)
4138 (COMMAND_DESCRIPTOR_t *)_LoadDataAddress,
4139 #endif // CC_Load
4140 #if (PAD_LIST || CC_Quote)
4141 (COMMAND_DESCRIPTOR_t *)_QuoteDataAddress,
4142 #endif // CC_Quote
4143 #if (PAD_LIST || CC_RSA_Decrypt)
4144 (COMMAND_DESCRIPTOR_t *)_RSA_DecryptDataAddress,
4145 #endif // CC_RSA_Decrypt
```c
#if (PAD_LIST)
    (COMMAND_DESCRIPTOR_t *)0,
#endif

#if (PAD_LIST || (CC_HMAC_Start || CC_MAC_Start))
    # if CC_HMAC_Start
        (COMMAND_DESCRIPTOR_t *)_HMAC_StartDataAddress,
    # endif
    # if CC_MAC_Start
        (COMMAND_DESCRIPTOR_t *)_MAC_StartDataAddress,
    # endif
    # if (CC_HMAC_Start || CC_MAC_Start) > 1
        # error "More than one aliased command defined"
    # endif
#endif

#if (PAD_LIST || CC_SequenceUpdate)
    (COMMAND_DESCRIPTOR_t *)_SequenceUpdateDataAddress,
#endif

#if (PAD_LIST || CC_Sign)
    (COMMAND_DESCRIPTOR_t *)_SignDataAddress,
#endif

#if (PAD_LIST || CC_Unseal)
    (COMMAND_DESCRIPTOR_t *)_UnsealDataAddress,
#endif

#if (PAD_LIST)
    (COMMAND_DESCRIPTOR_t *)0,
#endif

#if (PAD_LIST || CC_PolicySigned)
    (COMMAND_DESCRIPTOR_t *)_PolicySignedDataAddress,
#endif

#if (PAD_LIST || CC_ContextLoad)
    (COMMAND_DESCRIPTOR_t *)_ContextLoadDataAddress,
#endif

#if (PAD_LIST || CC_ContextSave)
    (COMMAND_DESCRIPTOR_t *)_ContextSaveDataAddress,
#endif

#if (PAD_LIST || CC_ECDH_KeyGen)
    (COMMAND_DESCRIPTOR_t *)_ECDH_KeyGenDataAddress,
#endif

#if (PAD_LIST || CC_EncryptDecrypt)
    (COMMAND_DESCRIPTOR_t *)_EncryptDecryptDataAddress,
#endif

#if (PAD_LIST || CC_FlushContext)
    (COMMAND_DESCRIPTOR_t *)_FlushContextDataAddress,
#endif

#if (PAD_LIST)
    (COMMAND_DESCRIPTOR_t *)0,
#endif

#if (PAD_LIST || CC_LoadExternal)
    (COMMAND_DESCRIPTOR_t *)_LoadExternalDataAddress,
#endif

#if (PAD_LIST || CC_MakeCredential)
    (COMMAND_DESCRIPTOR_t *)_MakeCredentialDataAddress,
#endif

#if (PAD_LIST || CC_NV_ReadPublic)
    (COMMAND_DESCRIPTOR_t *)_NV_ReadPublicDataAddress,
#endif

#if (PAD_LIST || CC_PolicyAuthorize)
    (COMMAND_DESCRIPTOR_t *)_PolicyAuthorizeDataAddress,
#endif

#if (PAD_LIST || CC_PolicyAuthValue)
    (COMMAND_DESCRIPTOR_t *)_PolicyAuthValueDataAddress,
#endif

#if (PAD_LIST || CC_PolicyCommandCode)
    (COMMAND_DESCRIPTOR_t *)_PolicyCommandCodeDataAddress,
#endif

#if (PAD_LIST || CC_PolicyCounterTimer)
```

---

4146  #if (PAD_LIST)
4147   (COMMAND_DESCRIPTOR_t *)0,
4148  #endif
4149  #if (PAD_LIST || (CC_HMAC_Start || CC_MAC_Start))
4150  # if CC_HMAC_Start
4151     (COMMAND_DESCRIPTOR_t *)_HMAC_StartDataAddress,
4152  # endif
4153  # if CC_MAC_Start
4154     (COMMAND_DESCRIPTOR_t *)_MAC_StartDataAddress,
4155  # endif
4156  # if (CC_HMAC_Start || CC_MAC_Start) > 1
4157     # error "More than one aliased command defined"
4158  # endif
4159  #endif // CC_HMAC_Start CC_MAC_Start
4160  #if (PAD_LIST || CC_SequenceUpdate)
4161     (COMMAND_DESCRIPTOR_t *)_SequenceUpdateDataAddress,
4162  #endif // CC_SequenceUpdate
4163  #if (PAD_LIST || CC_Sign)
4164     (COMMAND_DESCRIPTOR_t *)_SignDataAddress,
4165  #endif // CC_Sign
4166  #if (PAD_LIST || CC_Unseal)
4167     (COMMAND_DESCRIPTOR_t *)_UnsealDataAddress,
4168  #endif // CC_Unseal
4169  #if (PAD_LIST)
4170     (COMMAND_DESCRIPTOR_t *)0,
4171  #endif
4172  #if (PAD_LIST || CC_PolicySigned)
4173     (COMMAND_DESCRIPTOR_t *)_PolicySignedDataAddress,
4174  #endif // CC_PolicySigned
4175  #if (PAD_LIST || CC_ContextLoad)
4176     (COMMAND_DESCRIPTOR_t *)_ContextLoadDataAddress,
4177  #endif // CC_ContextLoad
4178  #if (PAD_LIST || CC_ContextSave)
4179     (COMMAND_DESCRIPTOR_t *)_ContextSaveDataAddress,
4180  #endif // CC_ContextSave
4181  #if (PAD_LIST || CC_ECDH_KeyGen)
4182     (COMMAND_DESCRIPTOR_t *)_ECDH_KeyGenDataAddress,
4183  #endif // CC_ECDH_KeyGen
4184  #if (PAD_LIST || CC_EncryptDecrypt)
4185     (COMMAND_DESCRIPTOR_t *)_EncryptDecryptDataAddress,
4186  #endif // CC_EncryptDecrypt
4187  #if (PAD_LIST || CC_FlushContext)
4188     (COMMAND_DESCRIPTOR_t *)_FlushContextDataAddress,
4189  #endif // CC_FlushContext
4190  #if (PAD_LIST)
4191     (COMMAND_DESCRIPTOR_t *)0,
4192  #endif
4193  #if (PAD_LIST || CC_LoadExternal)
4194     (COMMAND_DESCRIPTOR_t *)_LoadExternalDataAddress,
4195  #endif // CC_LoadExternal
4196  #if (PAD_LIST || CC_MakeCredential)
4197     (COMMAND_DESCRIPTOR_t *)_MakeCredentialDataAddress,
4198  #endif // CC_MakeCredential
4199  #if (PAD_LIST || CC_NV_ReadPublic)
4200     (COMMAND_DESCRIPTOR_t *)_NV_ReadPublicDataAddress,
4201  #endif // CC_NV_ReadPublic
4202  #if (PAD_LIST || CC_PolicyAuthorize)
4203     (COMMAND_DESCRIPTOR_t *)_PolicyAuthorizeDataAddress,
4204  #endif // CC_PolicyAuthorize
4205  #if (PAD_LIST || CC_PolicyAuthValue)
4206     (COMMAND_DESCRIPTOR_t *)_PolicyAuthValueDataAddress,
4207  #endif // CC_PolicyAuthValue
4208  #if (PAD_LIST || CC_PolicyCommandCode)
4209     (COMMAND_DESCRIPTOR_t *)_PolicyCommandCodeDataAddress,
4210  #endif // CC_PolicyCommandCode
4211  #if (PAD_LIST || CC_PolicyCounterTimer)
(COMMAND_DESCRIPTOR_t *)_PolicyCounterTimerDataAddress,
#endif // CC_PolicyCounterTimer
#if (PAD_LIST || CC_PolicyCpHash)
(COMMAND_DESCRIPTOR_t *)_PolicyCpHashDataAddress,
#endif // CC_PolicyCpHash
#if (PAD_LIST || CC_PolicyLocality)
(COMMAND_DESCRIPTOR_t *)_PolicyLocalityDataAddress,
#endif // CC_PolicyLocality
#if (PAD_LIST || CC_PolicyNameHash)
(COMMAND_DESCRIPTOR_t *)_PolicyNameHashDataAddress,
#endif // CC_PolicyNameHash
#if (PAD_LIST || CC_PolicyOR)
(COMMAND_DESCRIPTOR_t *)_PolicyORDataAddress,
#endif // CC_PolicyOR
#if (PAD_LIST || CC_PolicyTicket)
(COMMAND_DESCRIPTOR_t *)_PolicyTicketDataAddress,
#endif // CC_PolicyTicket
#if (PAD_LIST || CC_ReadPublic)
(COMMAND_DESCRIPTOR_t *)_ReadPublicDataAddress,
#endif // CC_ReadPublic
#if (PAD_LIST || CC_RSA_Encrypt)
(COMMAND_DESCRIPTOR_t *)_RSA_EncryptDataAddress,
#endif // CC_RSA_Encrypt
#if (PAD_LIST || CC_StartAuthSession)
(COMMAND_DESCRIPTOR_t *)_StartAuthSessionDataAddress,
#endif // CC_StartAuthSession
#if (PAD_LIST || CC_VerifySignature)
(COMMAND_DESCRIPTOR_t *)_VerifySignatureDataAddress,
#endif // CC_VerifySignature
#if (PAD_LIST || CC_ECC_Parameters)
(COMMAND_DESCRIPTOR_t *)_ECC_ParametersDataAddress,
#endif // CC_ECC_Parameters
#if (PAD_LIST || CC_FirmwareRead)
(COMMAND_DESCRIPTOR_t *)_FirmwareReadDataAddress,
#endif // CC_FirmwareRead
#if (PAD_LIST || CC_GetCapability)
(COMMAND_DESCRIPTOR_t *)_GetCapabilityDataAddress,
#endif // CC_GetCapability
#if (PAD_LIST || CC_GetRandom)
(COMMAND_DESCRIPTOR_t *)_GetRandomDataAddress,
#endif // CC_GetRandom
#if (PAD_LIST || CC_GetTestResult)
(COMMAND_DESCRIPTOR_t *)_GetTestResultDataAddress,
#endif // CC_GetTestResult
#if (PAD_LIST || CC_Hash)
(COMMAND_DESCRIPTOR_t *)_HashDataAddress,
#endif // CC_Hash
#if (PAD_LIST || CC_PCR_Read)
(COMMAND_DESCRIPTOR_t *)_PCR_ReadDataAddress,
#endif // CC_PCR_Read
#if (PAD_LIST || CC_PolicyPCR)
(COMMAND_DESCRIPTOR_t *)_PolicyPCRDataAddress,
#endif // CC_PolicyPCR
#if (PAD_LIST || CC_PolicyRestart)
(COMMAND_DESCRIPTOR_t *)_PolicyRestartDataAddress,
#endif // CC_PolicyRestart
#if (PAD_LIST || CC_ReadClock)
(COMMAND_DESCRIPTOR_t *)_ReadClockDataAddress,
#endif // CC_ReadClock
#if (PAD_LIST || CC_PCR_Extend)
(COMMAND_DESCRIPTOR_t *)_PCR_ExtendDataAddress,
#endif // CC_PCR_Extend
#if (PAD_LIST || CC_PCR_SetAuthValue)
(COMMAND_DESCRIPTOR_t *)_PCR_SetAuthValueDataAddress,
#endif // CC_PCR_SetAuthValue
#if (PAD_LIST || CC_NV_Certify)
(COMMAND_DESCRIPTOR_t *)_NV_CertifyDataAddress,
#endif // CC_NV_Certify
#if (PAD_LIST || CC_EventSequenceComplete)
(COMMAND_DESCRIPTOR_t *)_EventSequenceCompleteDataAddress,
#endif // CC_EventSequenceComplete
#if (PAD_LIST || CC_HashSequenceStart)
(COMMAND_DESCRIPTOR_t *)_HashSequenceStartDataAddress,
#endif // CC_HashSequenceStart
#if (PAD_LIST || CC_PolicyPhysicalPresence)
(COMMAND_DESCRIPTOR_t *)_PolicyPhysicalPresenceDataAddress,
#endif // CC_PolicyPhysicalPresence
#if (PAD_LIST || CC_PolicyDuplicationSelect)
(COMMAND_DESCRIPTOR_t *)_PolicyDuplicationSelectDataAddress,
#endif // CC_PolicyDuplicationSelect
#if (PAD_LIST || CC_PolicyPassword)
(COMMAND_DESCRIPTOR_t *)_PolicyPasswordDataAddress,
#endif // CC_PolicyPassword
#if (PAD_LIST || CC_ZGen_2Phase)
(COMMAND_DESCRIPTOR_t *)_ZGen_2PhaseDataAddress,
#endif // CC_ZGen_2Phase
#if (PAD_LIST || CC_EC_Ephemeral)
(COMMAND_DESCRIPTOR_t *)_EC_EphemeralDataAddress,
#endif // CC_EC_Ephemeral
#if (PAD_LIST || CC_PolicyNvWritten)
(COMMAND_DESCRIPTOR_t *)_PolicyNvWrittenDataAddress,
#endif // CC_PolicyNvWritten
#if (PAD_LIST || CC_PolicyTemplate)
(COMMAND_DESCRIPTOR_t *)_PolicyTemplateDataAddress,
#endif // CC_PolicyTemplate
#if (PAD_LIST || CC_CreateLoaded)
(COMMAND_DESCRIPTOR_t *)_CreateLoadedDataAddress,
#endif // CC_CreateLoaded
#if (PAD_LIST || CC_PolicyAuthorizeNV)
(COMMAND_DESCRIPTOR_t *)_PolicyAuthorizeNVDataAddress,
#endif // CC_PolicyAuthorizeNV
#if (PAD_LIST || CC_EncryptDecrypt2)
(COMMAND_DESCRIPTOR_t *)_EncryptDecrypt2DataAddress,
#endif // CC_EncryptDecrypt2
#if (PAD_LIST || CC_AC_GetCapability)
(COMMAND_DESCRIPTOR_t *)_AC_GetCapabilityDataAddress,
#endif // CC_AC_GetCapability
#if (PAD_LIST || CC_AC_Send)
(COMMAND_DESCRIPTOR_t *)_AC_SendDataAddress,
#endif // CC_AC_Send
#if (PAD_LIST || CC_Policy_AC_SendSelect)
(COMMAND_DESCRIPTOR_t *)_Policy_AC_SendSelectDataAddress,
#endif // CC_Policy_AC_SendSelect
#if (PAD_LIST || CC_Vendor_TCG_Test)
(COMMAND_DESCRIPTOR_t *)_Vendor_TCG_TestDataAddress,
#endif // CC_Vendor_TCG_Test
};
#endif // _COMMAND_TABLE_DISPATCH_
5.7 Commands.h

```c
#ifndef COMMANDS_H
#define COMMANDS_H

Start-up

#ifdef TPM_CC_Startup
#include "Startup_fp.h"
#endif

#ifdef TPM_CC_Shutdown
#include "Shutdown_fp.h"
#endif

Testing

#ifdef TPM_CC_SelfTest
#include "SelfTest_fp.h"
#endif

#ifdef TPM_CC_IncrementalSelfTest
#include "IncrementalSelfTest_fp.h"
#endif

#ifdef TPM_CC_GetTestResult
#include "GetTestResult_fp.h"
#endif

Session Commands

#ifdef TPM_CC_StartAuthSession
#include "StartAuthSession_fp.h"
#endif

#ifdef TPM_CC_PolicyRestart
#include "PolicyRestart_fp.h"
#endif

Object Commands

#ifdef TPM_CC_Create
#include "Create_fp.h"
#endif

#ifdef TPM_CC_Load
#include "Load_fp.h"
#endif

#ifdef TPM_CC_LoadExternal
#include "LoadExternal_fp.h"
#endif

#ifdef TPM_CC_ReadPublic
#include "ReadPublic_fp.h"
#endif

#ifdef TPM_CC_ActivateCredential
#include "ActivateCredential_fp.h"
#endif

#ifdef TPM_CC_MakeCredential
#include "MakeCredential_fp.h"
#endif

#ifdef TPM_CC_Unseal
#include "Unseal_fp.h"
#endif

#ifdef TPM_CC_ObjectChangeAuth
#include "ObjectChangeAuth_fp.h"
#endif

#ifdef TPM_CC_CreateLoaded
#include "CreateLoaded_fp.h"
#endif
```

Duplication Commands

```c
#ifdef TPM_CC_Duplicate
#include "Duplicate_fp.h"
#endif
#ifdef TPM_CC_Rewrap
#include "Rewrap_fp.h"
#endif
#ifdef TPM_CC_Import
#include "Import_fp.h"
#endif
```

Asymmetric Primitives

```c
#ifdef TPM_CC_RSA_Encrypt
#include "RSA_Encrypt_fp.h"
#endif
#ifdef TPM_CC_RSA_Decrypt
#include "RSA_Decrypt_fp.h"
#endif
#ifdef TPM_CC_ECDH_KeyGen
#include "ECDH_KeyGen_fp.h"
#endif
#ifdef TPM_CC_ECDH_ZGen
#include "ECDH_ZGen_fp.h"
#endif
#ifdef TPM_CC_ECC_Parameters
#include "ECC_Parameters_fp.h"
#endif
#ifdef TPM_CC_ZGen_2Phase
#include "ZGen_2Phase_fp.h"
#endif
```

Symmetric Primitives

```c
#ifdef TPM_CC_EncryptDecrypt
#include "EncryptDecrypt_fp.h"
#endif
#ifdef TPM_CC_EncryptDecrypt2
#include "EncryptDecrypt2_fp.h"
#endif
#ifdef TPM_CC_Hash
#include "Hash_fp.h"
#endif
#ifdef TPM_CC_HMAC
#include "HMAC_fp.h"
#endif
#ifdef TPM_CC_MAC
#include "MAC_fp.h"
#endif
```

Random Number Generator

```c
#ifdef TPM_CC_GetRandom
#include "GetRandom_fp.h"
#endif
#ifdef TPM_CC_StirRandom
#include "StirRandom_fp.h"
#endif
```

Hash/HMAC/Event Sequences
99  #ifdef TPM_CC_HMAC_Start
100  #include "HMAC_Start_fp.h"
101  #endif
102  #ifdef TPM_CC_MAC_Start
103  #include "MAC_Start_fp.h"
104  #endif
105  #ifdef TPM_CC_HashSequenceStart
106  #include "HashSequenceStart_fp.h"
107  #endif
108  #ifdef TPM_CC_SequenceUpdate
109  #include "SequenceUpdate_fp.h"
110  #endif
111  #ifdef TPM_CC_SequenceComplete
112  #include "SequenceComplete_fp.h"
113  #endif
114  #ifdef TPM_CC_EventSequenceComplete
115  #include "EventSequenceComplete_fp.h"
116  #endif

Attestation Commands

117  #ifdef TPM_CC_Certify
118  #include "Certify_fp.h"
119  #endif
120  #ifdef TPM_CC_CertifyCreation
121  #include "CertifyCreation_fp.h"
122  #endif
123  #ifdef TPM_CC_Qoute
124  #include "Quote_fp.h"
125  #endif
126  #ifdef TPM_CC_GetSessionAuditDigest
127  #include "GetSessionAuditDigest_fp.h"
128  #endif
129  #ifdef TPM_CC_GetCommandAuditDigest
130  #include "GetCommandAuditDigest_fp.h"
131  #endif
132  #ifdef TPM_CC_GetTime
133  #include "GetTime_fp.h"
134  #endif

Ephemeral EC Keys

135  #ifdef TPM_CC_Commit
136  #include "Commit_fp.h"
137  #endif
138  #ifdef TPM_CC_EC_Ephemeral
139  #include "EC_Ephemeral_fp.h"
140  #endif

Signing and Signature Verification

141  #ifdef TPM_CC_VerifySignature
142  #include "VerifySignature_fp.h"
143  #endif
144  #ifdef TPM_CC_Sign
145  #include "Sign_fp.h"
146  #endif

Command Audit

147  #ifdef TPM_CC_SetCommandCodeAuditStatus
148  #include "SetCommandCodeAuditStatus_fp.h"
149  #endif
Integrity Collection (PCR)

```c
#define TPM_CC_PCR_Extend
#include "PCR_Extend_fp.h"
#endif
#define TPM_CC_PCR_Event
#include "PCR_Event_fp.h"
#endif
#define TPM_CC_PCR_Read
#include "PCR_Read_fp.h"
#endif
#define TPM_CC_PCR_Allocate
#include "PCR_Allocate_fp.h"
#endif
#define TPM_CC_PCR_SetAuthPolicy
#include "PCR_SetAuthPolicy_fp.h"
#endif
#define TPM_CC_PCR_SetAuthValue
#include "PCR_SetAuthValue_fp.h"
#endif
#define TPM_CC_PCR_Reset
#include "PCR_Reset_fp.h"
#endif
```

Enhanced Authorization (EA) Commands

```c
#define TPM_CC_PolicySigned
#include "PolicySigned_fp.h"
#endif
#define TPM_CC_PolicySecret
#include "PolicySecret_fp.h"
#endif
#define TPM_CC_PolicyTicket
#include "PolicyTicket_fp.h"
#endif
#define TPM_CC_PolicyOR
#include "PolicyOR_fp.h"
#endif
#define TPM_CC_PolicyPCR
#include "PolicyPCR_fp.h"
#endif
#define TPM_CC_PolicyLocality
#include "PolicyLocality_fp.h"
#endif
#define TPM_CC_PolicyNV
#include "PolicyNV_fp.h"
#endif
#define TPM_CC_PolicyCounterTimer
#include "PolicyCounterTimer_fp.h"
#endif
#define TPM_CC_PolicyCommandCode
#include "PolicyCommandCode_fp.h"
#endif
#define TPM_CC_PolicyPhysicalPresence
#include "PolicyPhysicalPresence_fp.h"
#endif
#define TPM_CC_PolicyCpHash
#include "PolicyCpHash_fp.h"
#endif
#define TPM_CC_PolicyNameHash
#include "PolicyNameHash_fp.h"
#endif
#define TPM_CC_PolicyDuplicationSelect
#include "PolicyDuplicationSelect_fp.h"
#endif
```
#ifdef TPM_CC_PolicyAuthorize
#include "PolicyAuthorize_fp.h"
#endif

#include "PolicyAuthValue_fp.h"
#endif

#include "PolicyPassword_fp.h"
#endif

#include "PolicyGetDigest_fp.h"
#endif

#include "PolicyNvWritten_fp.h"
#endif

#include "PolicyTemplate_fp.h"
#endif

#include "PolicyAuthorizeNV_fp.h"
#endif

#include "CreatePrimary_fp.h"
#endif

#include "HierarchyControl_fp.h"
#endif

#include "SetPrimaryPolicy_fp.h"
#endif

#include "ChangePPS_fp.h"
#endif

#include "ChangeEPS_fp.h"
#endif

#include "Clear_fp.h"
#endif

#include "ClearControl_fp.h"
#endif

#include "HierarchyChangeAuth_fp.h"
#endif

#include "DictionaryAttackLockReset_fp.h"
#endif

#include "DictionaryAttackParameters_fp.h"
#endif

#include "PP_Commands_fp.h"
#endif

#include "SetAlgorithmSet_fp.h"
#endif
#include "SetAlgorithmSet_fp.h"
#endif

Field Upgrade

#include "FieldUpgradeStart_fp.h"
#endif
#include "FieldUpgradeData_fp.h"
#endif
#include "FirmwareRead_fp.h"
#endif

#include "ContextSave_fp.h"
#endif
#include "ContextLoad_fp.h"
#endif
#include "FlushContext_fp.h"
#endif
#include "EvictControl_fp.h"
#endif

#include "ReadClock_fp.h"
#endif
#include "ClockSet_fp.h"
#endif
#include "ClockRateAdjust_fp.h"
#endif

#include "GetCapability_fp.h"
#endif
#include "TestParms_fp.h"
#endif

#include "NV_DefineSpace_fp.h"
#endif
#include "NV_UndefineSpace_fp.h"
#endif
#include "NV_UndefineSpaceSpecial_fp.h"
#endif
#include "NV_ReadPublic"
#include "NV_ReadPublic_fp.h"
#endif

#ifdef TPM_CC_NV_Write
#include "NV_Write_fp.h"
#endif

define TPM_CC_NV_Increment
#include "NV_Increment_fp.h"
#endif

#define TPM_CC_NV_Extend
#include "NV_Extend_fp.h"
#endif

#define TPM_CC_NV_SetBits
#include "NV_SetBits_fp.h"
#endif

#define TPM_CC_NV_WriteLock
#include "NV_WriteLock_fp.h"
#endif

#define TPM_CC_NV_GlobalWriteLock
#include "NV_GlobalWriteLock_fp.h"
#endif

#define TPM_CC_NV_Read
#include "NV_Read_fp.h"
#endif

#define TPM_CC_NV_ReadLock
#include "NV_ReadLock_fp.h"
#endif

#define TPM_CC_NV_ChangeAuth
#include "NV_ChangeAuth_fp.h"
#endif

#define TPM_CC_NV_Certify
#include "NV_Certify_fp.h"
#endif

Attached Components

#define TPM_CC_AC_GetCapability
#include "AC_GetCapability_fp.h"
#endif

#define TPM_CC_AC_Send
#include "AC_Send_fp.h"
#endif

#define TPM_CC_Policy_AC_SendSelect
#include "Policy_AC_SendSelect_fp.h"
#endif

Vendor Specific

#define TPM_CC_Vendor_TCG_Test
#include "Vendor_TCG_Test_fp.h"
#endif

#define TPM_CC_Vendor_TCG_Test
#include "Vendor_TCG_Test_fp.h"
#endif
5.8 CompilerDependencies.h

This file contains the build switches. This contains switches for multiple versions of the crypto-library so some may not apply to your environment.

```c
#ifndef _COMPILER_DEPENDENCIES_H_
#define _COMPILER_DEPENDENCIES_H_

#ifdef GCC
  # define _MSC_VER
  # define WIN32
#endif

#ifdef _MSC_VER

These definitions are for the Microsoft compiler Endian conversion for aligned structures

#define REVERSE_ENDIAN_16(_Number) _byteswap_ushort(_Number)
#define REVERSE_ENDIAN_32(_Number) _byteswap_ulong(_Number)
#define REVERSE_ENDIAN_64(_Number) _byteswap_uint64(_Number)

Avoid compiler warning for in line of stdio (or not)

#define _NO_CRT_STDIO_INLINE

This macro is used to handle LIB_EXPORT of function and variable names in lieu of a .def file. Visual Studio requires that functions be explicitly exported and imported.

#define LIB_EXPORT __declspec(dllexport)

// VS compatible version
#define LIB_IMPORT __declspec(dllimport)

This is defined to indicate a function that does not return. Microsoft compilers do not support the _Noretrun() function parameter.

#define NORETURN __declspec(noreturn)

#ifdef _WIN64
#define _INTPTR 2
#else
#define _INTPTR 1
#endif

#define NOT_REFERENCED(x) (x)

Lower the compiler error warning for system include files. They tend not to be that clean and there is no reason to sort through all the spurious errors that they generate when the normal error level is set to /Wall

#define _REDUCE_WARNING_LEVEL_(n)
__pragma(warning(push, n))

#define _NORMAL_WARNING_LEVEL_
__pragma(warning(pop))

#include <stdint.h>

#ifndef _MSC_VER
#ifndef WINAPI
  #define WINAPI
#endif

#define __pragma(x)

#define REVERSE_ENDIAN_16(_Number) __builtin_bswap16(_Number)
```

# define REVERSE_ENDIAN_32(_Number) __builtin_bswap32(_Number)

#define REVERSE_ENDIAN_64(_Number) __builtin_bswap64(_Number)
#endif

#if defined(__GNUC__)
#define NORETURN __attribute__((noreturn))
#include <stdint.h>
#endif

Things that are not defined should be defined as NULL

#ifndef NORETURN
#define NORETURN
#endif
#ifndef LIB_EXPORT
#define LIB_EXPORT
#endif
#ifndef LIB_IMPORT
#define LIB_IMPORT
#endif
#ifndef _REDUCE_WARNING_LEVEL_
#define _REDUCE_WARNING_LEVEL_(n)
#endif
#ifndef _NORMAL_WARNING_LEVEL_
#define _NORMAL_WARNING_LEVEL_
#endif
#ifndef NOT_REFERENCED
#define NOT_REFERENCED(x) (x = x)
#endif
#ifndef _POSIX_
typedef int SOCKET;
#endif
#endif // _COMPILER_DEPENDENCIES_H_
5.9 Global.h

5.9.1 Description

This file contains internal global type definitions and data declarations that are need between subsystems. The instantiation of global data is in Global.c. The initialization of global data is in the subsystem that is the primary owner of the data.

The first part of this file has the typedefs for structures and other defines used in many portions of the code. After the typedef section, is a section that defines global values that are only present in RAM. The next three sections define the structures for the NV data areas: persistent, orderly, and state save. Additional sections define the data that is used in specific modules. That data is private to the module but is collected here to simplify the management of the instance data. All the data is instanced in Global.c.

```c
#if !defined _TPM_H_
#error "Should only be instanced in TPM.h"
#endif

5.9.2 Includes

#ifndef GLOBAL_H
#define GLOBAL_H

#include <string.h>
#include <stddef.h>

#include "Capabilities.h"
#include "TpmTypes.h"
#include "CommandAttributes.h"
#include "CryptTest.h"
#include "BnValues.h"
#include "CryptHash.h"
#include "CryptSym.h"
#include "CryptRsa.h"
#include "CryptEcc.h"
#include "CryptRand.h"
#include "CryptTest.h"
#include "TpmError.h"
#include "NV.h"

//** Defines and Types

//*** Crypto Self-Test Values
extern ALGORITHM_VECTOR g_implementedAlgorithms;
extern ALGORITHM_VECTOR g_toTest;

//*** Size Types
// These types are used to differentiate the two different size values used.

typedef UINT16 NUMBYTES;

//*** Other Types
// An AUTH_VALUE is a BYTE array containing a digest (TPMU_HA)
typedef BYTE AUTH_VALUE[sizeof(TPMU_HA)];
```
A TIME_INFO is a BYTE array that can contain a TPMS_TIME_INFO

```c
typedef BYTE TIME_INFO[sizeof(TPMS_TIME_INFO)];
```

A NAME is a BYTE array that can contain a TPMU_NAME

```c
typedef BYTE NAME[sizeof(TPMSU_NAME)];
```

Definition for a PROOF value

```c
TPM2B_TYPE(PROOF, PROOF_SIZE);
```

Definition for a Primary Seed value

```c
TPM2B_TYPE(SEED, PRIMARY_SEED_SIZE);
```

A CLOCK_NONCE is used to tag the time value in the authorization session and in the ticket computation so that the ticket expires when there is a time discontinuity. When the clock stops during normal operation, the nonce is 64-bit value kept in RAM but it is a 32-bit counter when the clock only stops during power events.

```c
#if CLOCK_STOPS
typedef UINT64 CLOCK_NONCE;
#else
typedef UINT32 CLOCK_NONCE;
#endif
```

### 5.9.3 Loaded Object Structures

#### 5.9.3.1 Description

The structures in this section define the object layout as it exists in TPM memory.

Two types of objects are defined: an ordinary object such as a key, and a sequence object that may be a hash, HMAC, or event.

#### 5.9.3.2 OBJECT_ATTRIBUTES

An OBJECT_ATTRIBUTES structure contains the variable attributes of an object. These properties are not part of the public properties but are used by the TPM in managing the object. An OBJECT_ATTRIBUTES is used in the definition of the OBJECT data type.

```c
typedef struct {
    unsigned publicOnly : 1;  //0) SET if only the public portion of
    unsigned epsHierarchy : 1; //1) SET if the object belongs to EPS
    unsigned ppsHierarchy : 1; //2) SET if the object belongs to PPS
    unsigned spsHierarchy : 1; //3) SET if the object belongs to SPS
    unsigned evict : 1;        //4) SET if the object is a platform or
                              //   owner evict object. Platform-
                              //   evict object belongs to SPS
                              //   hierarchy, owner-evict object
                              //   belongs to SPS or EPS hierarchy.
                              //   This bit is also used to mark a
                              //   completed sequence object so it
```
typedef struct OBJECT {
    // The attributes field is required to be first followed by the publicArea.
    // This allows the overlay of the object structure and a sequence structure
    OBJECT_ATTRIBUTES attributes;  // object attributes
    TPMT_PUBLIC publicArea;  // public area of an object
    TPMT_SENSITIVE sensitive;  // sensitive area of an object

    #if ALG_RSA
    TPM2B_NAME privateExponent;  // Additional field for the private
    #endif
    TPM2B_NAME qualifiedName;  // object qualified name
    TPMI_DH_OBJECT evictHandle;  // if the object is an evict object,
                                 // the original handle is kept here.
    // The 'working' handle will be the handle of an object slot.
    TPM2B_NAME name;  // Name of the object name. Kept here
                      // to avoid repeatedly computing it.
} OBJECT;

5.9.3.3 OBJECT Structure

An OBJECT structure holds the object public, sensitive, and meta-data associated. This structure is implementation dependent. For this implementation, the structure is not optimized for space but rather for clarity of the reference implementation. Other implementations may choose to overlap portions of the structure that are not used simultaneously. These changes would necessitate changes to the source code but those changes would be compatible with the reference implementation.

5.9.3.4 HASH_OBJECT Structure

This structure holds a hash sequence object or an event sequence object.

The first four components of this structure are manually set to be the same as the first four components of the object structure. This prevents the object from being inadvertently misused as sequence objects occupy the same memory as a regular object. A debug check is present to make sure that the offsets are what they are supposed to be.
typedef struct HASH_OBJECT
{
    OBJECT_ATTRIBUTES attributes; // The attributes of the HASH object
    TPMI_ALG_PUBLIC type; // algorithm
    TPMI_ALG_HASH nameAlg; // name algorithm
    TPMA_OBJECT objectAttributes; // object attributes
    // The data below is unique to a sequence object
    TPM2B_AUTH auth; // authorization for use of sequence
union
    {
        HASH_STATE hashState[HASH_COUNT];
        HMAC_STATE hmacState;
    }
    state;
} HASH_OBJECT;

typedef BYTE HASH_OBJECT_BUFFER[sizeof(HASH_OBJECT)];

5.9.3.5 ANY_OBJECT

This is the union for holding either a sequence object or a regular object. for ContextSave() and ContextLoad()
typedef union ANY_OBJECT
{
    OBJECT entity;
    HASH_OBJECT hash;
} ANY_OBJECT;
typedef BYTE ANY_OBJECT_BUFFER[sizeof(ANY_OBJECT)];

5.9.4 AUTH_DUP Types

These values are used in the authorization processing.
typedef UINT32 AUTH_ROLE;
#define AUTH_NONE ((AUTH_ROLE)(0))
#define AUTH_USER ((AUTH_ROLE)(1))
#define AUTH_ADMIN ((AUTH_ROLE)(2))
#define AUTH_DUP ((AUTH_ROLE)(3))

5.9.5 Active Session Context

5.9.5.1 Description

The structures in this section define the internal structure of a session context.

5.9.5.2 SESSION_ATTRIBUTES

The attributes in the SESSION_ATTRIBUTES structure track the various properties of the session. It maintains most of the tracking state information for the policy session. It is used within the SESSION structure.
typedef struct SESSION_ATTRIBUTES
{
    unsigned isPolicy : 1; //1) SET if the session may only be used
    // for policy
    unsigned isAudit : 1; //2) SET if the session is used for audit
    unsigned isBound : 1; //3) SET if the session is bound to with an
5.9.5.3 SESSION Structure

The SESSION structure contains all the context of a session except for the associated contextID.

NOTE: The contextID of a session is only relevant when the session context is stored off the TPM.
5.9.6 PCR

5.9.6.1 PCR_SAVE Structure

The PCR_SAVE structure type contains the PCR data that are saved across power cycles. Only the static PCR are required to be saved across power cycles. The DRTM and resettable PCR are not saved. The number of static and resettable PCR is determined by the platform-specific specification to which the TPM is built.
257  // This counter increments whenever the PCR are updated.  
258  // NOTE: A platform-specific specification may designate  
259  //      certain PCR changes as not causing this counter  
260  //      to increment.  
261  UINT32              pcrCounter;  
262 }) PCR_SAVE;  

5.9.6.2 PCR_POLICY  

263 #if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0  

This structure holds the PCR policies, one for each group of PCR controlled by policy.  

264 typedef struct PCR_POLICY  
265 {  
266   TPMI_ALG_HASH       hashAlg[NUM_POLICY_PCR_GROUP];  
267   TPM2B_DIGEST        a;  
268   TPM2B_DIGEST        policy[NUM_POLICY_PCR_GROUP];  
269 }) PCR_POLICY;  
270 #endif  

5.9.6.3 PCR_AUTHVALUE  

This structure holds the PCR policies, one for each group of PCR controlled by policy.  

271 typedef struct PCR_AUTH_VALUE  
272 {  
273   TPM2B_DIGEST        auth[NUM_AUTHVALUE_PCR_GROUP];  
274 }) PCR_AUTHVALUE;  

5.9.7 STARTUP_TYPE  

This enumeration is the possible startup types. The type is determined by the combination of  
TPM2_ShutDown() and TPM2_Startup().  

275 typedef enum  
276 {  
277   SU_RESET,  
278   SU_RESTART,  
279   SU_RESUME  
280 }) STARTUP_TYPE;  

5.9.8 NV  

5.9.8.1 NV_INDEX  

The NV_INDEX structure defines the internal format for an NV index. The indexData size varies  
according to the type of the index. In this implementation, all of the index is manipulated as a unit.  

281 typedef struct NV_INDEX  
282 {  
283   TPMS_NV_PUBLIC      publicArea;  
284   TPM2B_AUTH          authValue;  
285 }) NV_INDEX;
5.9.8.2 NV_REF

An NV_REF is an opaque value returned by the NV subsystem. It is used to reference and NV Index in a relatively efficient way. Rather than having to continually search for an Index, its reference value may be used. In this implementation, an NV_REF is a byte pointer that points to the copy of the NV memory that is kept in RAM.

```
286 typedef UINT32 NV_REF;
287 typedef BYTE *NV_RAM_REF;
```

5.9.8.3 NV_PIN

This structure deals with the possible endianess differences between the canonical form of the TPMS_NV_PIN_COUNTER_PARAMETERS structure and the internal value. The structures allow the data in a PIN index to be read as an 8-octet value using NvReadUINT64Data(). That function will byte swap all the values on a little endian system. This will put the bytes with the 4-octet values in the correct order but will swap the pinLimit and pinCount values. When written, the PIN index is simply handled as a normal index with the octets in canonical order.

```
288 #if BIG_ENDIAN_TPM
289 typedef struct
290 {
291     UINT32      pinCount;
292     UINT32      pinLimit;
293 } PIN_DATA;
294 #else
295 typedef struct
296 {
297     UINT32      pinLimit;
298     UINT32      pinCount;
299 } PIN_DATA;
300 #endif
301 typedef union
302 {
303     UINT64     intVal;
304     PIN_DATA   pin;
305 } NV_PIN;
```

5.9.9 COMMIT_INDEX_MASK

This is the define for the mask value that is used when manipulating the bits in the commit bit array. The commit counter is a 64-bit value and the low order bits are used to index the commitArray. This mask value is applied to the commit counter to extract the bit number in the array.

```
306 #if ALG_ECC
307 #define COMMIT_INDEX_MASK ((UINT16)((sizeof(gr.commitArray)*8)-1))
308 #endif
```

5.9.10 RAM Global Values

5.9.10.1 Description

The values in this section are only extant in RAM. They are defined here and instanced in Global.c.
5.9.10.2  g_rcIndex

This array is used to contain the array of values that are added to a return code when it is a parameter-, handle-, or session-related error. This is an implementation choice and the same result can be achieved by using a macro.

```c
extern const UINT16 g_rcIndex[15];
```

5.9.10.3  g_exclusiveAuditSession

This location holds the session handle for the current exclusive audit session. If there is no exclusive audit session, the location is set to TPM_RH_UNASSIGNED.

```c
extern TPM_HANDLE g_exclusiveAuditSession;
```

5.9.10.4  g_time

This is the value in which we keep the current command time. This is initialized at the start of each command. The time is the accumulated time since the last time that the TPM's timer was last powered up. Clock is the accumulated time since the last time that the TPM was cleared. g_time is in mS.

```c
extern UINT64 g_time;
```

5.9.10.5  g_timeEpoch

This value contains the current clock Epoch. It changes when there is a clock discontinuity. It may be necessary to place this in NV should the timer be able to run across a power down of the TPM but not in all cases (e.g. dead battery). If the nonce is placed in NV, it should go in gp because it should be changing slowly.

```c
#if CLOCK_STOPS
extern CLOCK_NONCE g_timeEpoch;
#else
#define g_timeEpoch gp.timeEpoch
#endif
```

5.9.10.6  g_phEnable

This is the platform hierarchy control and determines if the platform hierarchy is available. This value is SET on each TPM2_Startup(). The default value is SET.

```c
extern BOOL g_phEnable;
```

5.9.10.7  g_pcrReConfig

This value is SET if a TPM2_PCR_Allocate() command successfully executed since the last TPM2_Startup(). If so, then the next shutdown is required to be Shutdown(CLEAR).

```c
extern BOOL g_pcrReConfig;
```

5.9.10.8  g_DRTMHandle

This location indicates the sequence object handle that holds the DRTM sequence data. When not used, it is set to TPM_RH_UNASSIGNED. A sequence DRTM sequence is started on either _TPM_Init() or _TPM_Hash_Start().
5.9.10.9  g_DrtmPreStartup

This value indicates that an H-CRTM occurred after _TPM_Init() but before TPM2_Startup(). The define for PRE_STARTUP_FLAG is used to add the g_DrtmPreStartup value to gp.orderlyState at shutdown. This hack is to avoid adding another NV variable.

5.9.10.10 g_StartupLocality3

This value indicates that a TPM2_Startup() occurred at locality 3. Otherwise, it at locality 0. The define for STARTUP_LOCALITY_3 is to indicate that the startup was not at locality 0. This hack is to avoid adding another NV variable.

5.9.10.11 TPM_SU_NONE

Part 2 defines the two shutdown/startup types that may be used in TPM2_Shutdown() and TPM2_Startup(). This additional define is used by the TPM to indicate that no shutdown was received.

NOTE: This is a reserved value.

5.9.10.12 TPM_SU_DA_USED

As with TPM_SU_NONE, this value is added to allow indication that the shutdown was not orderly and that a DA=protected object was reference during the previous cycle.

5.9.10.13 Startup Flags

These flags are included in gp.orderlyState. These are hacks and are being used to avoid having to change the layout of gp. The PRE_STARTUP_FLAG indicates that a _TPM_Hash_Start()/Data()/End() sequence was received after _TPM_Init() but before TPM2_StartUp(). STARTUP_LOCALITY_3 indicates that the last TPM2_Startup() was received at locality 3. These flags are only relevant if after a TPM2_Shutdown(STATE).

5.9.10.14 g_daUsed

This location indicates if a DA-protected value is accessed during a boot cycle. If none has, then there is no need to increment failedTries on the next non-orderly startup. This bit is merged with gp.orderly when that gp.orderly is set to SU_NONE_VALUE
5.9.10.15  g_updateNV

This flag indicates if NV should be updated at the end of a command. This flag is set to UT_NONE at the beginning of each command in ExecuteCommand(). This flag is checked in ExecuteCommand() after the detailed actions of a command complete. If the command execution was successful and this flag is not UT_NONE, any pending NV writes will be committed to NV. UT_ORDERLY causes any RAM data to be written to the orderly space for staging the write to NV.

5.9.10.16  g_powerWasLost

This flag is used to indicate if the power was lost. It is SET in _TPM__Init(). This flag is cleared by TPM2_Startup() after all power-lost activities are completed.

NOTE: When power is applied, this value can come up as anything. However, _plat__WasPowerLost() will provide the proper indication in that case. So, when power is actually lost, we get the correct answer. When power was not lost, but the power-lost processing has not been completed before the next _TPM_Init(), then the TPM still does the correct thing.

5.9.10.17  g_clearOrderly

This flag indicates if the execution of a command should cause the orderly state to be cleared. This flag is set to FALSE at the beginning of each command in ExecuteCommand() and is checked in ExecuteCommand() after the detailed actions of a command complete but before the check of g_updateNV. If this flag is TRUE, and the orderly state is not SU_NONE_VALUE, then the orderly state in NV memory will be changed to SU_NONE_VALUE or SU_DA_USED_VALUE.

5.9.10.18  g_prevOrderlyState

This location indicates how the TPM was shut down before the most recent TPM2_Startup(). This value, along with the startup type, determines if the TPM should do a TPM Reset, TPM Restart, or TPM Resume.

5.9.10.19  g_nvOk

This value indicates if the NV integrity check was successful or not. If not and the failure was severe, then the TPM would have been put into failure mode after it had been re-manufactured. If the NV failure was in the area where the state-save data is kept, then this variable will have a value of FALSE indicating that a TPM2_Startup(CLEAR) is required.
NV availability is sampled as the start of each command and stored here so that its value remains consistent during the command execution

```c
extern TPM_RC g_NvStatus;
```

### 5.9.10.20 g_platformUnique

This location contains the unique value(s) used to identify the TPM. It is loaded on every `_TPM2_Startup()` The first value is used to seed the RNG. The second value is used as a vendor `authValue`. The value used by the RNG would be the value derived from the chip unique value (such as fused) with a dependency on the authorities of the code in the TPM boot path. The second would be derived from the chip unique value with a dependency on the details of the code in the boot path. That is, the first value depends on the various signers of the code and the second depends on what was signed. The TPM vendor should not be able to know the first value but they are expected to know the second.

```c
extern TPM2B_AUTH g_platformUniqueAuthorities; // Reserved for RNG
extern TPM2B_AUTH g_platformUniqueDetails; // referenced by VENDOR_PERMANENT
```

### 5.9.11 Persistent Global Values

#### 5.9.11.1 Description

The values in this section are global values that are persistent across power events. The lifetime of the values determines the structure in which the value is placed.

#### 5.9.11.2 PERSISTENT_DATA

This structure holds the persistent values that only change as a consequence of a specific Protected Capability and are not affected by TPM power events (TPM2_Startup() or TPM2_Shutdown()).

```c
typedef struct {
    // Hierarchy authPolicies
    TPMI_ALG_HASH ownerAlg;
    TPMI_ALG_HASH endorsementAlg;
    TPMI_ALG_HASH lockoutAlg;
    TPM2B_DIGEST ownerPolicy;
    TPM2B_DIGEST endorsementPolicy;
    TPM2B_DIGEST lockoutPolicy;
    // Primary Seeds
    TPM2B_SEED EPSeed;
    TPM2B_SEED SPSeed;
    TPM2B_SEED PPSeed;
    // Note there is a nullSeed in the state_reset memory.
}
```
372    // Hierarchy proofs
373    TPM2B_PROOF          phProof;
374    TPM2B_PROOF          shProof;
375    TPM2B_PROOF          ehProof;
376    // Note there is a nullProof in the state_reset memory.
377
378    //***********************************************************************
379    // Reset Events
380    //***********************************************************************
381    // A count that increments at each TPM reset and never get reset during the life
382    // time of TPM. The value of this counter is initialized to 1 during TPM
383    // manufacture process. It is used to invalidate all saved contexts after a TPM
384    // Reset.
385    UINT64              totalResetCount;
386
387    // This counter increments on each TPM Reset. The counter is reset by
388    // TPM2_Clear().
389    UINT32              resetCount;
390
391    //***********************************************************************
392    // PCR
393    //***********************************************************************
394    // This structure hold the policies for those PCR that have an update policy.
395    // This implementation only supports a single group of PCR controlled by
396    // policy. If more are required, then this structure would be changed to
397    // an array.
398    #if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
399    PCR_POLICY          pcrPolicies;
400    #endif
401
402    // This structure indicates the allocation of PCR. The structure contains a
403    // list of PCR allocations for each implemented algorithm. If no PCR are
404    // allocated for an algorithm, a list entry still exists but the bit map
405    // will contain no SET bits.
406    TPML_PCR_SELECTION  pcrAllocated;
407
408    //***********************************************************************
409    // Physical Presence
410    //***********************************************************************
411    // The PP_LIST type contains a bit map of the commands that require physical
412    // to be asserted when the authorization is evaluated. Physical presence will be
413    // checked if the corresponding bit in the array is SET and if the authorization
414    // handle is TPM_RH_PLATFORM.
415    //
416    // These bits may be changed with TPM2_PP_Commands().
417    BYTE                ppList[(COMMAND_COUNT + 7) / 8];
418
419    //***********************************************************************
420    // Dictionary attack values
421    //***********************************************************************
422    // These values are used for dictionary attack tracking and control.
423    UINT32              failedTries;       // the current count of unexpired
424    // authorization failures
425
426    UINT32              maxTries;         // number of unexpired authorization
427    // failures before the TPM is in
428    // lockout
429
430    UINT32              recoveryTime;      // time between authorization failures
431    // before failedTries is decremented
432
433    UINT32              lockoutRecovery;   // time that must expire between
434    // authorization failures associated
435    // with lockoutAuth
436
437    BOOL                lockOutAuthEnabled; // TRUE if use of lockoutAuth is
/**
 * Orderly State
 */

TPM_SU orderlyState;

//*****************************************************************************
// Command audit values.
//*****************************************************************************
BYTE auditCommands[((COMMAND_COUNT + 1) + 7) / 8];

TPMI_ALG_HASH auditHashAlg;

UINT64 auditCounter;

//*****************************************************************************
// Algorithm selection
//*****************************************************************************

UINT32 algorithmSet;

//*****************************************************************************
// Firmware version
//*****************************************************************************

UINT32 firmwareV1;

UINT32 firmwareV2;

//*****************************************************************************
// Timer Epoch
//*****************************************************************************

CLOCK_NONCE timeEpoch;

#define CLOCK_STOPS

5.9.11.3 ORDERLY_DATA

The data in this structure is saved to NV on each TPM2_Shutdown().
// Clock has two parts. One is the state save part and one is the NV part. The
// state save version is updated on each command. When the clock rolls over, the
// NV version is updated. When the TPM starts up, if the TPM was shutdown in and
// orderly way, then the sClock value is used to initialize the clock. If the
// TPM shutdown was not orderly, then the persistent value is used and the safe
// attribute is clear.

UINT64 clock; // The orderly version of clock
TPMI_YES_NO clockSafe; // Indicates if the clock value is
// safe.

// In many implementations, the quality of the entropy available is not that
// high. To compensate, the current value of the drbgState can be saved and
// restored on each power cycle. This prevents the internal state from reverting
// to the initial state on each power cycle and starting with a limited amount
// of entropy. By keeping the old state and adding entropy, the entropy will
// accumulate.

DRBG_STATE drbgState;

// These values allow the accumulation of self-healing time across orderly shutdown
// of the TPM.

#if ACCUMULATE_SELF_HEAL_TIMER
UINT64 selfHealTimer; // current value of s_selfHealTime
UINT64 lockoutTimer; // current value of s_lockoutTime
UINT64 time; // current value of g_time at shutdown
#endif // ACCUMULATE_SELF_HEAL_TIMER

ORDERLY_DATA go;

5.9.11.4 STATE_CLEAR_DATA

This structure contains the data that is saved on Shutdown(STATE). and restored on Startup(STATE). The values are set to their default settings on any Startup(Clear). In other words the data is only persistent across TPM Resume.

If the comments associated with a parameter indicate a default reset value, the value is applied on each Startup(CLEAR).

typedef struct state_clear_data
{
    //***************************************************************************
    //               Hierarchy Control
    //***************************************************************************
    BOOL shEnable; // default reset is SET
    BOOL ehEnable; // default reset is SET
    BOOL phEnableNV; // default reset is SET
    TPMI_ALG_HASH platformAlg; // default reset is TPM_ALG_NULL
    TPM2B_DIGEST platformPolicy; // default reset is an Empty Buffer
    TPM2B_DIGEST platformAuth; // default reset is an Empty Buffer

    //***************************************************************************
    //               PCR
    //***************************************************************************
    PCR pcrSave; // default reset is 0...0

    // This structure hold the authorization values for those PCR that have an

5.9.11.5 State Reset Data

This structure contains data that is saved on Shutdown(STATE) and restored on the subsequent Startup(ANY). That is, the data is preserved across TPM Resume and TPM Restart.

If a default value is specified in the comments this value is applied on TPM Reset.

```c
typedef struct state_reset_data
{
    // Hierarchy Control
    TPM2B_PROOF nullProof;  // The proof value associated with
    // the TPM_RH_NULL hierarchy. The
    // default reset value is from the RNG.

    TPM2B_SEED nullSeed;  // The seed value for the TPM_RN_NULL
    // hierarchy. The default reset value
    // is from the RNG.

    // Context
    UINT32 clearCount;  // The default reset value is 0.

    UINT64 objectContextID;  // This is the context ID for a saved
    // object context. The default reset
    // value is 0.

    #ifndef NDEBUG
    #undef CONTEXT_SLOT
    #define CONTEXT_SLOT BYTE
    #endif

    CONTEXT_SLOT contextArray[MAX_ACTIVE_SESSIONS];  // This array contains
    // the version numbers of saved
    // contexts (see Session.c in for details). The
    // default reset value is {0}.

    CONTEXT_COUNTER contextCounter;  // This is the value from which the
    // 'contextID' is derived. The
    // default reset value is {0}.

    // Command Audit
    // When an audited command completes, ExecuteCommand() checks the return
    // value. If it is TPM_RC_SUCCESS, and the command is an audited command,
    // TPM will extend the cpHash and rpHash for the command to this value. If this
    // digest was the Zero Digest before the cpHash was extended, the audit counter
```
// is incremented.

TPM2B_DIGEST commandAuditDigest; // This value is set to an Empty Digest by TPM2_GetCommandAuditDigest() or a TPM Reset.

//****************************************************************************
// Boot counter
//****************************************************************************

UINT32 restartCount; // This counter counts TPM Restarts.
// The default reset value is 0.

//****************************************************************************
// PCR
//****************************************************************************

// This counter increments whenever the PCR are updated. This counter is preserved across TPM Resume even though the PCR are not preserved. This is because sessions remain active across TPM Restart and the count value in the session as long as the sessions are active.
// NOTE: A platform-specific specification may designate that certain PCR changes do not increment this counter to increment.

UINT32 pcrCounter; // The default reset value is 0.

#if ALG_ECC

//****************************************************************************
// ECDAA
//****************************************************************************

UINT64 commitCounter; // This counter increments each time TPM2_Commit() returns TPM_RC_SUCCESS. The default reset value is 0.

TPM2B_NONCE commitNonce; // This random value is used to compute the commit values. The default reset value is from the RNG.

// This implementation relies on the number of bits in g_commitArray being a power of 2 (8, 16, 32, 64, etc.) and no greater than 64K

BYTE commitArray[16]; // The default reset value is {0}.

#endif // ALG_ECC

) STATE_RESET_DATA;

extern STATE_RESET_DATA gr;

5.9.12 NV Layout

The NV data organization is:
a) a PERSISTENT_DATA structure
b) a STATE_RESET_DATA structure
c) a STATE_CLEAR_DATA structure
d) an ORDERLY_DATA structure
e) the user defined NV index space

#define NV_PERSISTENT_DATA  (0)
#define NV_STATE_RESET_DATA (NV_PERSISTENT_DATA + sizeof(PERSISTENT_DATA))
#define NV_STATE_CLEAR_DATA (NV_STATE_RESET_DATA + sizeof(STATE_RESET_DATA))
#define NV_ORDERLY_DATA (NV_STATE_CLEAR_DATA + sizeof(STATE_CLEAR_DATA))
#define NV_INDEX_RAM_DATA (NV_ORDERLY_DATA + sizeof(ORDERLY_DATA))
#define NV_USER_DYNAMIC     (NV_INDEX_RAM_DATA + sizeof(s_indexOrderlyRam))
#define NV_USER_DYNAMIC_END   _NV_MEMORY_SIZE

## 5.9.13 Global Macro Definitions

The NV_READ_PERSISTENT and NV_WRITE_PERSISTENT macros are used to access members of the PERSISTENT_DATA structure in NV.

```c
#define NV_READ_PERSISTENT(to, from)  
    NvRead(&to, offsetof(PERSISTENT_DATA, from), sizeof(to))
#define NV_WRITE_PERSISTENT(to, from)  
    NvWrite(offsetof(PERSISTENT_DATA, to), sizeof(gp.to), &from)
#define CLEAR_PERSISTENT(item)    
    NvClearPersistent(offsetof(PERSISTENT_DATA, item), sizeof(gp.item))
#define NV_SYNC_PERSISTENT(item)  
    NV_WRITE_PERSISTENT(item, gp.item)
```

At the start of command processing, the index of the command is determined. This index value is used to access the various data tables that contain per-command information. There are multiple options for how the per-command tables can be implemented. This is resolved in GetClosestCommandIndex().

```c
typedef UINT16 COMMAND_INDEX;
#define UNIMPLEMENTED_COMMAND_INDEX   ((COMMAND_INDEX)(~0))
typedef struct _COMMAND_FLAGS_
{
    unsigned trialPolicy : 1;  //1) If SET, one of the handles references a
    //   trial policy and authorization may be
    //   skipped. This is only allowed for a policy
    //   command.
} COMMAND_FLAGS;
```

This structure is used to avoid having to manage a large number of parameters being passed through various levels of the command input processing.

```c
typedef struct _COMMAND_
{
    TPM_ST tag;           // the parsed command tag
    TPM_CC code;          // the parsed command code
    COMMAND_INDEX index;  // the computed command index
    UINT32 handleNum;     // the number of entity handles in the
    // handle area of the command
    TPM_HANDLE handles[MAX_HANDLE_NUM];  // the parsed handle values
    UINT32 sessionNum;    // the number of sessions found
    INT32 parameterSize;  // starts out with the parsed command size
    // and is reduced and values are
    // unmarshaled. Just before calling the
    // command actions, this should be zero.
    // After the command actions, this number
    // should grow as values are marshaled
    // in to the response buffer.
    // this is initialized with the parsed size
    // of authorizationSize field and should
    // be zero when the authorizations are
    // parsed.
    BYTE *parameterBuffer; // input to ExecuteCommand
    BYTE *responseBuffer;  // input to ExecuteCommand

#if ALG_SHA1
    TPM2B_SHA1_DIGEST sha1CpHash;
    TPM2B_SHA1_DIGEST sha1RpHash;
#endif
#if ALG_SHA256
    TPM2B_SHA256_DIGEST sha256CpHash;
    TPM2B_SHA256_DIGEST sha256RpHash;
#endif
```
707  #if ALG_SHA384
708   TPM2B_SHA384_DIGEST sha384CpHash;
709   TPM2B_SHA384_DIGEST sha384RpHash;
710  #endif
711  #if ALG_SHA512
712   TPM2B_SHA512_DIGEST sha512CpHash;
713   TPM2B_SHA512_DIGEST sha512RpHash;
714  #endif
715  #if ALG_SM3_256
716   TPM2B_SM3_256_DIGEST sm3_256CpHash;
717   TPM2B_SM3_256_DIGEST sm3_256RpHash;
718  #endif
719 }
720 } COMMAND;
721
722 Global sting constants for consistency in KDF function calls.
723
724 extern const TPM2B *PRIMARY_OBJECT_CREATION;
725 extern const TPM2B *SECRET_KEY;
726 extern const TPM2B *SESSION_KEY;
727 extern const TPM2B *INTEGRITY_KEY;
728 extern const TPM2B *CONTEXT_KEY;
729 extern const TPM2B *STORAGE_KEY;
730 extern const TPM2B *SESSION_KEY;
731 extern const TPM2B *STORAGE_KEY;
732 #if defined SESSION_PROCESS_C || defined GLOBAL_C || defined MANUFACTURE_C
733    extern const TPM2B *IDENTITY_STRING;
734  // SELF_TEST
735  #endif
736// SELF_TEST
737 from Manufacture.c
738  extern BOOL g_manufactured;
739
740 This value indicates if a TPM2_Startup() commands has been receive since the power on event. This
741 flag is maintained in power simulation module because this is the only place that may reliably set this flag
742 to FALSE.
743
to FALSE.
744 extern BOOL g_initialized;
745
746 5.9.14 Private data
747
748 #if defined SESSION_PROCESS_C || defined GLOBAL_C || defined MANUFACTURE_C
749 From SessionProcess.c
750 The following arrays are used to save command sessions information so that the command
751 handle/session buffer does not have to be preserved for the duration of the command. These arrays are
752 indexed by the session index in accordance with the order of sessions in the session area of the
753 command.
754
755 Array of the authorization session handles
756 extern TPM_HANDLE s_sessionHandles[MAX_SESSION_NUM];
757
758 Array of authorization session attributes
759 extern TPMA_SESSION s_attributes[MAX_SESSION_NUM];
Array of handles authorized by the corresponding authorization sessions; and if none, then TPM_RH_UNASSIGNED value is used

```c
extern TPM_HANDLE s_associatedHandles[MAX_SESSION_NUM];
```

Array of nonces provided by the caller for the corresponding sessions

```c
extern TPM2B_NONCE s_nonceCaller[MAX_SESSION_NUM];
```

Array of authorization values (HMAC’s or passwords) for the corresponding sessions

```c
extern TPM2B_AUTH s_inputAuthValues[MAX_SESSION_NUM];
```

Array of pointers to the SESSION structures for the sessions in a command

```c
extern SESSION *s_usedSessions[MAX_SESSION_NUM];
```

Special value to indicate an undefined session index

```c
#define UNDEFINED_INDEX 0xFFFF
```

Index of the session used for encryption of a response parameter

```c
extern UINT32 s_encryptSessionIndex;
```

Index of the session used for decryption of a command parameter

```c
extern UINT32 s_decryptSessionIndex;
```

Index of a session used for audit

```c
extern UINT32 s_auditSessionIndex;
```

The cpHash for command audit

```c
#ifdef TPM_CC_GetCommandAuditDigest
extern TPM2B_DIGEST s_cpHashForCommandAudit;
#endif
```

Flag indicating if NV update is pending for the lockOutAuthEnabled or failedTries DA parameter

```c
extern BOOL s_DAPendingOnNV;
```

From DA.c

This variable holds the accumulated time since the last time that failedTries was decremented. This value is in millisecond.

```c
#if !ACCUMULATE_SELF_HEAL_TIMER
extern UINT64 s_selfHealTimer;
#endif // ACCUMULATE_SELF_HEAL_TIMER
```

This variable holds the accumulated time that the lockoutAuth has been blocked.

```c
extern UINT64 s_lockoutTimer;
```

```c
#endif // DA_C
```

```c
#if defined DA_C || defined GLOBAL_C || defined MANUFACTURE_C
```
From NV.c

This marks the end of the NV area. This is a run-time variable as it might not be compile-time constant.

```
760 extern NV_REF s_evictNvEnd;
```

This space is used to hold the index data for an orderly Index. It also contains the attributes for the index.

```
761 extern BYTE s_indexOrderlyRam[RAM_INDEX_SPACE]; // The orderly NV Index data
```

This value contains the current max counter value. It is written to the end of allocatable NV space each time an index is deleted or added. This value is initialized on Startup. The indices are searched and the maximum of all the current counter indices and this value is the initial value for this.

```
762 extern UINT64 s_maxCounter;
```

This is space used for the NV Index cache. As with a persistent object, the contents of a referenced index are copied into the cache so that the NV Index memory scanning and data copying can be reduced. Only code that operates on NV Index data should use this cache directly. When that action code runs, s_lastNvIndex will contain the index header information. It will have been loaded when the handles were verified.

**NOTE:** An NV index handle can appear in many commands that do not operate on the NV data (e.g. TPM2_StartAuthSession()). However, only one NV Index at a time is ever directly referenced by any command. If that changes, then the NV Index caching needs to be changed to accommodate that. Currently, the code will verify that only one NV Index is referenced by the handles of the command.

```
763 extern NV_INDEX s_cachedNvIndex;
764 extern NV_REF sCachedNvRef;
765 extern BYTE *s_cachedNvRamRef;
```

Initial NV Index/evict object iterator value

```
766 #define NV_REF_INIT (NV_REF)0xFFFFFFFF
767 #endif
768 #if defined OBJECT_C || defined GLOBAL_C
```

From Object.c

This type is the container for an object.

```
769 extern OBJECT s_objects[MAX_LOADEDOBJECTS];
770 #endif // OBJECT_C
771 #if defined PCR_C || defined GLOBAL_C
```

From PCR.c

```
772 typedef struct
773 {
774 #if ALG_SHA1
775   // SHA1 PCR
776   BYTE sha1Pcr[SHA1_DIGEST_SIZE];
777 #endif
778 #if ALG_SHA256
779   // SHA256 PCR
780   BYTE sha256Pcr[SHA256_DIGEST_SIZE];
781 #endif
782 #if ALG_SHA384
783   // SHA384 PCR
784   BYTE sha384Pcr[SHA384_DIGEST_SIZE];
785 #endif
786 #if ALG_SHA512
```
```c
787    // SHA512 PCR
788    BYTE    sha512Pcr[SHA512_DIGEST_SIZE];
789 #endif
790 #if    ALG_SM3_256
791    // SHA256 PCR
792    BYTE    sm3_256Pcr[SM3_256_DIGEST_SIZE];
793 #endif
794 } PCR;
795 typedef struct
796 {
797     unsigned int    stateSave : 1;
798     // if the PCR value should be
799     unsigned int    resetLocality : 5;
800     // The locality that the PCR
801     unsigned int    extendLocality : 5;
802     // The locality that the PCR
803     // can be extend
804 } PCR_Attributes;
805 extern PCR          s_pcrs[IMPLEMENTATION_PCR];
806 #if defined SESSION_C || defined GLOBAL_C
807 #if defined IO_BUFFER_C || defined GLOBAL_C
808     // Container for HMAC or policy session tracking information
809     typedef struct
810 {
811         BOOL             occupied;
812         SESSION         session;   // session structure
813     } SESSION_SLOT;
814     extern SESSION_SLOT     s_sessions[MAX_LOADED_SESSIONS];
815 
816     extern UINT32    s_oldestSavedSession;
817     The index in conextArray that has the value of the oldest saved session context. When no context is
818     saved, this will have a value that is greater than or equal to MAX_ACTIVE_SESSIONS.
819 
820     extern int         s_freeSessionSlots;
821 #endif // SESSION_C
822 #if defined IO_BUFFER_C || defined GLOBAL_C
823 #if defined MEMORY_LIB_C
824     // Each command function is allowed a structure for the inputs to the function and a structure for the
825     // outputs. The command dispatch code unmarshals the input butter to the command action input structure
826     // starting at the first byte of s_actionIoBuffer. The value of s_actionIoAllocation is the number of UINT64
827     // values allocated. It is used to set the pointer for the response structure. The command dispatch code will
828     // marshal the response values into the final output buffer.
829     extern UINT64    s_actionIoBuffer[768];    // action I/O buffer
830     extern UINT32    s_actionIoAllocation;    // number of UINT64 allocated for the
831     extern UINT32    s_actionIoInput;
832 #endif // MEMORY_LIB_C
833 From TPMFail.c
834     This value holds the address of the string containing the name of the function in which the failure
835     occurred. This address value isn't useful for anything other than helping the vendor to know in which file
836     the failure occurred.
```
Trusted Platform Module Library

Part 4: Supporting Routines

821 extern BOOL g_inFailureMode; // Indicates that the TPM is in failure mode
822 #if SIMULATION
823 extern BOOL g_forceFailureMode; // flag to force failure mode during test
824 #endif
825 typedef void (FailFunction)(const char *function, int line, int code);
826 #if defined TPM_FAIL_C || defined GLOBAL_C || 1
827 extern UINT32 s_failFunction;
828 extern UINT32 s_failLine; // the line in the file at which
829 // the error was signaled
830 extern UINT32 s_failCode; // the error code used
831 extern FailFunction *LibFailCallback;
832 #endif // TPM_FAIL_C

From CommandCodeAttributes.c

833 extern const TPMA_CC s_ccAttr[];
834 extern const COMMAND_ATTRIBUTES s_commandAttributes[];
835 #endif // GLOBAL_H
5.10 GpMacros.h

5.10.1 Introduction

This file is a collection of miscellaneous macros.

```c
#ifndef GP_MACROS_H
#define GP_MACROS_H
#ifndef NULL
#define NULL 0
#endif
#include "swap.h"
#include "VendorString.h"
#endif // GP_MACROS_H
```

5.10.2 For Self-test

These macros are used in CryptUtil() to invoke the incremental self test.

```c
#if SELF_TEST
#define TEST(alg) if(TEST_BIT(alg, g_toTest)) CryptTestAlgorithm(alg, NULL)
#else
#define TEST(alg) (alg)
#define TEST_HASH(alg) (alg)
#endif // SELF_TEST
```

5.10.3 For Failures

If implementation is using longjmp, then the call to TpmFail() does not return and the compiler will complain about unreachable code that comes after. To allow for not having longjmp, TpmFail() will return and the subsequent code will be executed. This macro accounts for the difference.

```c
#ifndef NO_LONGJMP
#define FAIL_RETURN(returnCode) (returnCode)
#else
#define FAIL_RETURN(returnCode) (returnCode)
#define TPM_FAIL_RETURN() return (returnCode)
#endif
```
This macro tests that a condition is TRUE and puts the TPM into failure mode if it is not. If longjmp is being used, then the FAIL(FATAL_ERROR_) macro makes a call from which there is no return. Otherwise, it returns and the function will exit with the appropriate return code.

```c
#define REQUIRE(condition, errorCode, returnCode)  
   {  
      if (!!condition)  
      {  
         FAIL(FATAL_ERROR_errorCode);  
         FAIL_RETURN(returnCode);  
      }  
   }
#define PARAMETER_CHECK(condition, returnCode)  
   REQUIRE((condition), PARAMETER, returnCode)
#define pAssert(a)  ((void)0)
#define pAssert(a) { if (!(a)) FAIL(FATAL_ERROR_PARAMETER);}
#endif
```

### 5.10.4 Derived from Vendor-specific values

Values derived from vendor specific settings in Implementation.h

```c
#define PCR_SELECT_MIN ((PLATFORM_PCR+7)/8)
#define PCR_SELECT_MAX ((IMPLEMENTATION_PCR+7)/8)
#define MAX_ORDERLY_COUNT ((1 << ORDERLY_BITS) - 1)
#define PRIVATE_VENDOR_SPECIFIC_BYTES ((MAX_RSA_KEY_BYTES/2) * (3 + CRT_FORMAT_RSA * 2))
```

### 5.10.5 Compile-time Checks

In some cases, the relationship between two values may be dependent on things that change based on various selections like the chosen cryptographic libraries. It is possible that these selections will result in incompatible settings. These are often detectable by the compiler but it isn’t always possible to do the check in the preprocessor code. For example, when the check requires use of `sizeof` then the preprocessor can’t do the comparison. For these cases, we include a special macro that, depending on the compiler will generate a warning to indicate if the check always passes or always fails because it involves fixed constants. To run these checks, define COMPILER_CHECKS in TpmBuildSwitches.h

```c
#define cAssert pAssert
#define cAssert(value)
```

This is used commonly in the Crypt code as a way to keep listings from getting too long. This is not to save paper but to allow one to see more useful stuff on the screen at any given time.

```c
#define ERROR_RETURN(returnCode)  
   {  
      retVal = returnCode;  
      goto Exit;  
   }
#define MAX(a, b) ((a) > (b) ? (a) : (b))
#define MIN(a, b) ((a) < (b) ? (a) : (b))
```
These are defined for use when the size of the vector being checked is known at compile time.

The following definitions are used if they have not already been defined. The defaults for these settings are compatible with ISO/IEC 9899:2011 (E)

If CONTEXT_INTEGRITY_HASH_ALG is defined, then the vendor is using the old style table. Otherwise, pick the strongest implemented hash algorithm as the context hash.
Unless some algorithm is broken...

This is the size that was used before the 1.38 errata requiring that P1.14.4 be followed

As required by P1.14.4

As required by P1.14.3.1

This is the pre-errata version

If CONTEXT_ENCRY_ALG is defined, then the vendor is using the old style table

This is updated to follow the requirement of P2 that the label not be larger than 32 bytes.
This bit is used to indicate that an authorization ticket expires on TPM Reset and TPM Restart. It is added to the timeout value returned by TPM2_PolicySigned() and TPM2_PolicySecret() and used by TPM2_PolicyTicket(). The timeout value is relative to Time (g_time). Time is reset whenever the TPM loses power and cannot be moved forward by the user (as can Clock). g_time is a 64-bit value expressing time in ms. Sealing the MSb() for a flag means that the TPM needs to be reset at least once every 292,471,208 years rather than once every 584,942,417 years.

```c
#define EXPIRATION_BIT ((UINT64)1 << 63)
```

Check for consistency of the bit ordering of bit fields

```c
#if BIG_ENDIAN_TPM && MOST_SIGNIFICANT_BIT_0 && USE_BIT_FIELD_STRUCTURES
  #error "Settings not consistent"
#endif
```

These macros are used to handle the variation in handling of bit fields. If

```c
#if USE_BIT_FIELD_STRUCTURES // The default, old version, with bit fields
  # define IS_ATTRIBUTE(a, type, b)    ((a.b != 0))
  # define SET_ATTRIBUTE(a, type, b)   (a.b = SET)
  # define CLEAR_ATTRIBUTE(a, type, b) (a.b = CLEAR)
  # define GET_ATTRIBUTE(a, type, b)   (a.b)
#else
  # define IS_ATTRIBUTE(a, type, b)         ((a & type##_##_b) != 0)
  # define SET_ATTRIBUTE(a, type, b)        (a |= type##_##_b)
  # define CLEAR_ATTRIBUTE(a, type, b)      (a &= ~type##_##_b)
  # define GET_ATTRIBUTE(a, type, b)        (type)((a & type##_##_b) >> type##_##_b##_SHIFT)
#endif
```

```c
#endif // GP_MACROS_H
```
5.11 InternalRoutines.h

```c
#ifndef INTERNAL_ROUTINES_H
#define INTERNAL_ROUTINES_H

#if !defined _LIB_SUPPORT_H_ && !defined _TPM_H_
#error "Should not be called"
#endif

DRTM functions

#include "_TPM_Hash_Start_fp.h"
#include "_TPM_Hash_Data_fp.h"
#include "_TPM_Hash_End_fp.h"

Internal subsystem functions

#include "Object_fp.h"
#include "Context_spt_fp.h"
#include "Object_spt_fp.h"
#include "Entity_fp.h"
#include "Session_fp.h"
#include "Hierarchy_fp.h"
#include "NvReserved_fp.h"
#include "NV_spt_fp.h"
#include "PCR_fp.h"
#include "DA_fp.h"
#include "TpmFail_fp.h"
#include "SessionProcess_fp.h"

Internal support functions

#include "CommandCodeAttributes_fp.h"
#include "Marshal_fp.h"
#include "Time_fp.h"
#include "Locality_fp.h"
#include "PP_fp.h"
#include "CommandAudit_fp.h"
#include "Manufacture_fp.h"
#include "Handle_fp.h"
#include "Power_fp.h"
#include "Response_fp.h"
#include "CommandDispatcher_fp.h"
#ifndef CC_AC_Send
#   include "AC_spt_fp.h"
#endif // CC_AC_Send

Miscellaneous

#include "Bits_fp.h"
#include "AlgorithmCap_fp.h"
#include "PropertyCap_fp.h"
#include "IoBuffers_fp.h"
#include "Memory_fp.h"
#include "ResponseCodeProcessing_fp.h"

Internal cryptographic functions

#include "BnConvert_fp.h"
#include "BnMath_fp.h"
#include "BnMemory_fp.h"
#include "Ticket_fp.h"
```
#include "CryptUtil_fp.h"
#include "CryptHash_fp.h"
#include "CryptSym_fp.h"
#include "CryptDes_fp.h"
#include "CryptPrime_fp.h"
#include "CryptRand_fp.h"
#include "CryptSelfTest_fp.h"
#include "MathOnByteBuffers_fp.h"
#include "CryptSym_fp.h"
#include "AlgorithmTests_fp.h"
#if ALG_RSA
#include "CryptRsa_fp.h"
#include "CryptPrimeSieve_fp.h"
#endif
#if ALG_ECC
#include "CryptEccMain_fp.h"
#include "CryptEccSignature_fp.h"
#include "CryptEccKeyExchange_fp.h"
#endif
#if CC_MAC || CC_MAC_Seed
    #include "CryptSmac_fp.h"
    #if ALG_CMAC
        #include "CryptCmac_fp.h"
    #endif
#endif

Support library
#include "SupportLibraryFunctionPrototypes_fp.h"

Linkage to platform functions
#include "Platform_fp.h"
#endif
5.12 LibSupport.h

This header file is used to select the library code that gets included in the TPM built

```c
#ifndef __LIB_SUPPORT_H__
#define __LIB_SUPPORT_H__

OSSL has a full suite but yields an executable that is much larger than it needs to be.

#define OSSL 1

LTC has symmetric support, RSA support, and inadequate ECC support

#define LTC 2

MSBN only provides math support so should not be used as the hash or symmetric library

#define MSBN 3

SYMCRYPT only provides symmetric cryptography so would need to be combined with another library that has math support

#define SYMCRYPT 4

#if RADIX_BITS == 32
#define RADIX_BYTES 4
#elif RADIX_BITS == 64
#define RADIX_BYTES 8
#else
#error "RADIX_BITS must either be 32 or 64."
#endif

#include the options for hashing If all the optional headers were always part of the distribution then it would not be necessary to do the conditional testing before the include. )-;

#if HASH_LIB == OSSL
#include "ossl/TpmToOsslHash.h"
#elif HASH_LIB == LTC
#include "ltc/TpmToLtcHash.h"
#elif HASH_LIB == SYMCRYPT
#include "symcrypt/TpmToSymcryptHash.h"
#else
#error "No hash library selected"
#endif

Set the linkage for the selected symmetric library

#if SYM_LIB == OSSL
#include "ossl/TpmToOsslSym.h"
#elif SYM_LIB == LTC
#include "ltc/TpmToLtcSym.h"
#elif SYM_LIB == SYMCRYPT
#include "symcrypt/TpmToSymcryptSym.h"
#else
#error "No symmetric library selected"
#endif

#define MIN

Select a big number Library. This uses a define rather than an include so that the header will not be included until the required values have been defined.
```
#if MATH_LIB == OSSL
#define MATHLIB_H "ossl/TpmToOsslMath.h"
#elif MATH_LIB == LTC
#define MATHLIB_H "ltc/TpmToLtcMath.h"
#elif MATH_LIB == MSBN
#define MATHLIB_H "msbn/TpmToMsBnMath.h"
#else
#error "No math library selected"
#endif
#endif // _LIB_SUPPORT_H_
5.13 NV.h

5.13.1 Index Type Definitions

These definitions allow the same code to be used pre and post 1.21. The main action is to redefine the
index type values from the bit values. Use TPM_NT_ORDINARY to indicate if the TPM_NT type is
defined.

```c
#ifndef _NV_H_
#define _NV_H_

#if TPM_NT_ORDINARY
    #define GET_TPM_NT(attributes) GET_ATTRIBUTE(attributes, TPMA_NV, TPM_NT)
#else
    #define GetNv_TPM_NV(attributes)
        (   IS_ATTRIBUTE(attributes, TPMA_NV, COUNTER)
            +   (IS_ATTRIBUTE(attributes, TPMA_NV, BITS) << 1)
            +   (IS_ATTRIBUTE(attributes, TPMA_NV, EXTEND) << 2)
        )
#endif

#define TPM_NT_ORDINARY (0)
#define TPM_NT_COUNTER (1)
#define TPM_NT_BITS (2)
#define TPM_NT_EXTEND (4)
#endif
```

5.13.2 Attribute Macros

These macros are used to isolate the differences in the way that the index type changed in version 1.21
of the specification.

```c
#define IsNvOrdinaryIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_ORDINARY)
#define IsNvCounterIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_COUNTER)
#define IsNvBitsIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_BITS)
#define IsNvExtendIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_EXTEND)
#endif
#define IsNvPinPassIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_PIN_PASS)
#endif
#define IsNvPinFailIndex(attributes) (GET_TPM_NT(attributes) == TPM_NT_PIN_FAIL)
```

```c
typedef struct {
    UINT32 size;
    TPM_HANDLE handle;
} NV_ENTRY_HEADER;

#define NV_EVICT_OBJECT_SIZE
    (sizeof(UINT32) + sizeof(TPM_HANDLE) + sizeof(OBJECT))
#define NV_INDEX_COUNTER_SIZE
    (sizeof(UINT32) + sizeof(NV_INDEX) + sizeof(UINT64))
#define NV_RAM_INDEX_COUNTER_SIZE
    (sizeof(NV_RAM_HEADER) + sizeof(UINT64))
```
typedef struct {
    UINT32 size;
    TPM_HANDLE handle;
    TPMA_NV attributes;
} NV_RAM_HEADER;

Defines the end-of-list marker for NV. The list terminator is a UINT32 of zero, followed by the current value of \texttt{s\_maxCounter} which is a 64-bit value. The structure is defined as an array of 3 UINT32 values so that there is no padding between the UINT32 list end marker and the UIT64m() \texttt{maxCounter} value.

typedef UINT32 NV_LIST_TERMINATOR[3];

5.13.3 Orderly RAM Values

The following defines are for accessing orderly RAM values. This is the initialize for the RAM reference iterator.

#define NV_RAM_REF_INIT 0

This is the starting address of the RAM space used for orderly data

#define RAM_ORDERLY_START \(~(s\_indexOrderlyRam[0])\)

This is the offset within NV that is used to save the orderly data on an orderly shutdown.

#define NV_ORDERLY_START \(NV\_INDEX\_RAM\_DATA)\)

This is the end of the orderly RAM space. It is actually the first byte after the last byte of orderly RAM data

#define RAM_ORDERLY_END \(RAM\_ORDERLY\_START + \text{sizeof}(s\_indexOrderlyRam)\)

This is the end of the orderly space in NV memory. As with \texttt{RAM\_ORDERLY\_END}, it is actually the offset of the first byte after the end of the NV orderly data.

#define NV_ORDERLY_END \(NV\_ORDERLY\_START + \text{sizeof}(s\_indexOrderlyRam)\)

Macro to check that an orderly RAM address is with range.

#define ORDERLY_RAM_ADDRESS_OK(start, offset) \((\text{start} >= RAM\_ORDERLY\_START) && ((\text{start} + \text{offset} - 1) < RAM\_ORDERLY\_END)\)

#define RETURN_IF_NV_IS_NOT_AVAILABLE \{
    \text{if}(g\_NvStatus != TPM\_RC\_SUCCESS) \{
        return g\_NvStatus;
    \}
\}

Routinely have to clear the orderly flag and fail if the NV is not available so that it can be cleared.

#define RETURN_IF_ORDERLY \{
    \text{if}(NvClearOrderly() != TPM\_RC\_SUCCESS) \{
        return g\_NvStatus;
    \}
\}

#define NV\_IS\_AVAILABLE \((g\_NvStatus == TPM\_RC\_SUCCESS)\)

#define IS\_ORDERLY(value) \((value < SU\_DA\_USED\_VALUE)\)

#define NV\_IS\_ORDERLY \(IS\_ORDERLY(gp\_.orderlyState)\)
Macro to set the NV UPDATE_TYPE. This deals with the fact that the update is possibly a combination of UT_NV and UT_ORDERLY.

```c
#define SET_NV_UPDATE(type)     g_updateNV |= (type)
#endif // _NV_H_
```
5.14 PRNG_TestVectors.h

```c
#define _MSBN_DRBG_TEST_VECTORS_H
#include "PRNG_TestVectors.h"

#define DRBG_ALGORITHM == TPM_ALG_AES && DRBG_KEY_BITS == 256
#define DRBG_KEY_SIZE_BITS == 256

#define DRBG_TEST_INITIATE_ENTROPY
0x0d, 0x15, 0xaa, 0x80, 0xb1, 0x6c, 0x3a, 0x10, \
0x90, 0x6c, 0xfe, 0xdb, 0x79, 0x5d, 0xae, 0x0b, \
0x5b, 0x81, 0x04, 0x1c, 0x5c, 0x5b, 0xfa, 0xcb, \
0x37, 0x3d, 0x44, 0x40, 0xd9, 0x12, 0x0f, 0x7e, \
0x3d, 0x6c, 0xf9, 0x09, 0x86, 0xcf, 0x52, 0xd8, \
0x5d, 0x3e, 0x94, 0x7d, 0x8c, 0x06, 0x1f, 0x91

#define DRBG_TEST_RESEED_ENTROPY
0x6e, 0xe7, 0x93, 0x39, 0x55, 0xfd, 0x2a, \
0x1, 0x73, 0xcb, 0x45, 0x9f, 0x3b, 0x60, 0x0d, \
0xad, 0x87, 0x09, 0x55, 0xf2, 0x2d, 0xa8, 0x0a, \
0xf8

#define DRBG_TEST_GENERATED_INTERM
0x28, 0xe0, 0xe9, 0xe8, 0x21, 0x01, 0x66, 0x50, \
0x8c, 0x8f, 0x65, 0xf2, 0x20, 0x7b, 0xda, 0xe0

#define DRBG_TEST_GENERATED
0x94, 0x6f, 0x51, 0x82, 0xd5, 0x45, 0x10, 0xb9, \
0x46, 0x12, 0x48, 0xf5, 0x71, 0xca, 0x06, 0xc9

#define DRBG_TEST_INITIATE_ENTROPY
0x8f, 0xc1, 0x1b, 0xdb, 0x5a, 0xab, 0xb7, 0xe0, \
0x93, 0xb6, 0x14, 0x28, 0xe0, 0x90, 0x73, 0x03, \
0xcb, 0x45, 0x9f, 0x3b, 0x60, 0x0d, 0xad, 0x87, \
0x09, 0x55, 0xf2, 0x2d, 0xa8, 0x0a, 0x44, 0xf8

#define DRBG_TEST_GENERATED_INTERM
0xc0, 0xd5, 0x3c, 0xd5, 0xec, 0xcd, 0x5a, 0x10, \
0xd7, 0xea, 0x26, 0x61, 0x11, 0x25, 0x9b, 0x05, \
0x57, 0x4f, 0xc6, 0xd8, 0xe8, 0xd8, 0xeb, \
0x72, 0x37, 0x8c, 0xf8, 0x2f, 0x1d, 0xba, 0x2a

#define DRBG_TEST_RESEED_ENTROPY
0x0d, 0x15, 0xaa, 0x80, 0xb1, 0x6c, 0x3a, 0x10, \
0x90, 0x6c, 0xfe, 0xdb, 0x79, 0x5d, 0xae, 0x0b, \
0x5b, 0x81, 0x04, 0x1c, 0x5c, 0x5b, 0xfa, 0xcb, \
0x37, 0x3d, 0x44, 0x40, 0xd9, 0x12, 0x0f, 0x7e, \
0x3d, 0x6c, 0xf9, 0x09, 0x86, 0xcf, 0x52, 0xd8, \
0x5d, 0x3e, 0x94, 0x7d, 0x8c, 0x06, 0x1f, 0x91

#define DRBG_TEST_GENERATED
0x28, 0xe0, 0xe9, 0xe8, 0x21, 0x01, 0x66, 0x50, \
0x8c, 0x8f, 0x65, 0xf2, 0x20, 0x7b, 0xda, 0xe0

#endif
#endif
```

Entropy is the size of the state. The state is the size of the key plus the IV. The IV is a block. Key = 256, block = 128, state = 384
5.15 SelfTest.h

5.15.1 Introduction

This file contains the structure definitions for the self-test. It also contains macros for use when the self-test is implemented.

```c
#ifndef _SELF_TEST_H_
#define _SELF_TEST_H_

5.15.2 Defines

Was typing this a lot

#define SELF_TEST_FAILURE   FAIL(FATAL_ERROR_SELF_TEST)

Use the definition of key sizes to set algorithm values for key size. Need to do this to avoid a lot of #ifdefs in the code. Also, define the index for each of the algorithms.

```c
#if ALG_AES && defined AES_KEY_SIZE_BITS_128
#define AES_128     YES
#define AES_128_INDEX   0
#else
#define AES_128     NO
#endif

#if ALG_AES && defined AES_KEY_SIZE_BITS_192
#define AES_192     YES
#define AES_192_INDEX   (AES_128)
#else
#define AES_192     NO
#endif

#if ALG_AES && defined AES_KEY_SIZE_BITS_256
#define AES_256     YES
#define AES_256_INDEX   (AES_128 + AES_192)
#else
#define AES_256     NO
#endif

#define SM4_128     YES
#define SM4_128_INDEX   (AES_128 + AES_192 + AES_256)
#else
#define SM4_128     NO
#endif

#define NUM_SYMS    (AES_128 + AES_192 + AES_256 + SM4_128)
typedef UINT32      SYM_INDEX;
```

These two defines deal with the fact that the TPM_ALG_ID table does not delimit the symmetric mode values with a TPM_SYM_MODE_FIRST and TPM_SYM_MODE_LAST

```c
#define TPM_SYM_MODE_FIRST       ALG_CTR_VALUE
#define TPM_SYM_MODE_LAST        ALG_ECB_VALUE
#define NUM_SYM_MODES   (TPM_SYM_MODE_LAST - TPM_SYM_MODE_FIRST + 1)
```

Define a type to hold a bit vector for the modes.

```c
#define NUM_SYM_MODES <= 0
#error "No symmetric modes implemented"
define NUM_SYM_MODES <= 8
typedef BYTE    SYM_MODES;
define NUM_SYM_MODES <= 16
```
typedef UINT16 SYM_MODES;

#elif NUM_SYM_MODES <= 32
typedef UINT32 SYM_MODES;
#else
#error "Too many symmetric modes"
#endif

typedef struct {
    const TPM_ALG_ID alg;     // the algorithm
    const UINT16 keyBits;     // bits in the key
    const BYTE *key;          // The test key
    const UINT32 ivSize;      // block size of the algorithm
    const UINT32 dataInOutSize; // size to encrypt/decrypt
    const BYTE *dataIn;       // data to encrypt
    const BYTE *dataOut[NUM_SYM_MODES]; // data to decrypt
} SYMMETRIC_TEST_VECTOR;

#if ALG_RSA
extern const RSA_KEY c_rsaTestKey; // This is a constant structure
#endif

#define SYM_TEST_VALUE_REF(value, alg, keyBits, mode) SIZED_REFERENCE(value##_##alg##keyBits##_##mode)

typedef struct {
    TPM_ALG_ID alg;
    UINT16 keySizeBits;
} SYM_ALG;

#define SET_ALG(ALG, v) MemorySetBit((v), ALG, sizeof(v) * 8)

#if ALG_SHA512
    #define DEFAULT_TEST_HASH ALG_SHA512_VALUE
    #define DEFAULT_TEST_DIGEST_SIZE SHA512_DIGEST_SIZE
    #define DEFAULT_TEST_HASH_BLOCK_SIZE SHA512_BLOCK_SIZE
#elif ALG_SHA384
    #define DEFAULT_TEST_HASH ALG_SHA384_VALUE
    #define DEFAULT_TEST_DIGEST_SIZE SHA384_DIGEST_SIZE
    #define DEFAULT_TEST_HASH_BLOCK_SIZE SHA384_BLOCK_SIZE
#elif ALG_SHA256
    #define DEFAULT_TEST_HASH ALG_SHA256_VALUE
    #define DEFAULT_TEST_DIGEST_SIZE SHA256_DIGEST_SIZE
    #define DEFAULT_TEST_HASH_BLOCK_SIZE SHA256_BLOCK_SIZE
#elif ALG_SHA1
    #define DEFAULT_TEST_HASH ALG_SHA1_VALUE
    #define DEFAULT_TEST_DIGEST_SIZE SHA1_DIGEST_SIZE
    #define DEFAULT_TEST_HASH_BLOCK_SIZE SHA1_BLOCK_SIZE
#endif

#endif // _SELF_TEST_H_
5.16 SupportLibraryFunctionPrototypes_fp.h

5.16.1 Introduction

This file contains the function prototypes for the functions that need to be present in the selected match library. For each function listed, there should be a small stub function. That stub provides the interface between the TPM code and the support library. In most cases, the stub function will only need to do a format conversion between the TPM big number and the support library big number. The TPM big number format was chosen to make this relatively simple and fast.

Arithmetic operations return a BOOL to indicate if the operation completed successfully or not.

```c
#ifndef SUPPORT_LIBRARY_FUNCTION_PROTOTYPES_H
#define SUPPORT_LIBRARY_FUNCTION_PROTOTYPES_H

5.16.2 SupportLibInit()

This function is called by CryptInit() so that necessary initializations can be performed on the cryptographic library.

LIB_EXPORT int SupportLibInit(void);

5.16.3 MathLibraryCompatibiltiyCheck()

This function is only used during development to make sure that the library that is being referenced is using the same size of data structures as the TPM.

LIB_EXPORT BOOL BnModMult(bigNum result, bigConst op1, bigConst op2, bigConst modulus);

LIB_EXPORT BOOL BnMult(bigNum result, bigConst multiplicand, bigConst multiplier);

LIB_EXPORT BOOL BnDiv(bigNum quotient, bigNum remainder, bigConst dividend, bigConst divisor);
```
5.16.7 BnMod()

```c
#define BnMod(a, b) BnDiv(NULL, (a), (a), (b))
```

5.16.8 BnGcd()

Get the greatest common divisor of two numbers. This function is only needed when the TPM implements RSA.

```c
LIB_EXPORT BOOL BnGcd(bigNum gcd, bigConst number1, bigConst number2);
```

5.16.9 BnModExp()

Do modular exponentiation using bigNum values. This function is only needed when the TPM implements RSA.

```c
LIB_EXPORT BOOL BnModExp(bigNum result, bigConst number, bigConst exponent, bigConst modulus);
```

5.16.10 BnModInverse()

Modular multiplicative inverse. This function is only needed when the TPM implements RSA.

```c
LIB_EXPORT BOOL BnModInverse(bigNum result, bigConst number, bigConst modulus);
```

5.16.11 BnEccModMult()

This function does a point multiply of the form R = [d]S. A return of FALSE indicates that the result was the point at infinity. This function is only needed if the TPM supports ECC.

```c
LIB_EXPORT BOOL BnEccModMult(bigPoint R, pointConst S, bigConst d, bigCurve E);
```

5.16.12 BnEccModMult2()

This function does a point multiply of the form R = [d]S + [u]Q. A return of FALSE indicates that the result was the point at infinity. This function is only needed if the TPM supports ECC.

```c
LIB_EXPORT BOOL BnEccModMult2(bigPoint R, pointConst S, bigConst d, pointConst Q, bigConst u, bigCurve E);
```

5.16.13 BnEccAdd()

This function does a point add R = S + Q. A return of FALSE indicates that the result was the point at infinity. This function is only needed if the TPM supports ECC.

```c
LIB_EXPORT BOOL BnEccAdd(bigPoint R, pointConst S, pointConst Q, bigCurve E);
```
5.16.14 BnCurveInitialize()

This function is used to initialize the pointers of a $bnCurve_t$ structure. The structure is a set of pointers to $bigNum$ values. The curve-dependent values are set by a different function. This function is only needed if the TPM supports ECC.

```c
LIB_EXPORT bigCurve
BnCurveInitialize(bigCurve E, TPM_ECC_CURVE curveId);
#endif
```
5.17 TPMB.h

This file contains extra TPM2B structures

```c
#ifndef _TPMB_H
#define _TPMB_H

TPM2B Types

typedef struct {
  UINT16 size;
  BYTE buffer[1];
} TPM2B, *P2B;

typedef const TPM2B *PC2B;

This macro helps avoid having to type in the structure in order to create a new TPM2B type that is used in a function.

#define TPM2B_TYPE(name, bytes)

typedef union {
  struct {
    UINT16 size;
    BYTE buffer[(bytes)];
  } t;
  TPM2B b;
} TPM2B_##name

This macro defines a TPM2B with a constant character value. This macro sets the size of the string to the size minus the terminating zero byte. This lets the user of the label add their terminating 0. This method is chosen so that existing code that provides a label will continue to work correctly.

#define TPM2B_STRING(name, value)

static const union {
  struct {
    UINT16 size;
    BYTE buffer[sizeof(value)];
  } t;
  TPM2B b;
} name##_ = {
  sizeof(value), {value}};

const TPM2B *name = &name##_b

Macro to instance and initialize a TPM2B value

#define TPM2B_INIT(TYPE, name)

  TPM2B_##TYPE name = {
    sizeof(name.t.buffer), {0}}

#define TPM2B_BYTE_VALUE(bytes) TPM2B_TYPE(bytes##_BYTE_VALUE, bytes)

#endif
```
5.18  Tpm.h

Root header file for building any TPM.lib code

```c
#elsedef _TPM_H_
#define _TPM_H_
#include "Implementation.h"  // Types from the library. These need to come before
#include "LibSupport.h"      // Global.h because some of the structures in
                          // that file depend on the structures used by the
                          // cryptographic libraries.
#include "GpMacros.h"        // Define additional macros
#include "Global.h"           // Define other TPM types
#include "InternalRoutines.h" // Function prototypes
#endif // _TPM_H_
```
5.19 TpmBuildSwitches.h

This file contains the build switches. This contains switches for multiple versions of the crypto-library so some may not apply to your environment.

The switches are guarded so that they can either be set on the command line or set here. If the switch is listed on the command line (-DSOME_SWITCH) with no setting, then the switch will be set to YES. If the switch setting is not on the command line or if the setting is other than YES or NO, then the switch will be set to the default value. The default can either be YES or NO as indicated on each line where the default is selected.

A caution. Do not try to test these macros by inserting #defines in this file. For some curious reason, a variable set on the command line with no setting will have a value of 1. An #if SOME_VARIABLE will work if the variable is not defined or is defined on the command line with no initial setting. However, a

```
//      #define SOME_VARIABLE
```

is a null string and when used in #if SOME_VARIABLE will not be a proper expression. If you want to test various switches, either use the command line or change the default.

```
#ifndef _TPM_BUILD_SWITCHES_H_
#define _TPM_BUILD_SWITCHES_H_

#define YES 1
#define NO 0

#define DEBUG   YES // Default: Either YES or NO
#endif
```

Need an unambiguous definition for DEBUG. Don't change this

```
#ifndef USE_BN_ECC_DATA
  |
  || ((USE_BN_ECC_DATA != NO) && (USE_BN_ECC_DATA != YES))
#define USE_BN_ECC_DATA     YES // Default: Either YES or NO
#endif
```

The SIMULATION switch allows certain other macros to be enabled. The things that can be enabled in a simulation include key caching, reproducible random sequences, instrumentation of the RSA key generation process, and certain other debug code. SIMULATION Needs to be defined as either YES or NO. This grouping of macros will make sure that it is set correctly. A simulated TPM would include a Virtual TPM. The interfaces for a Virtual TPM should be modified from the standard ones in the Simulator project.

If SIMULATION is in the compile parameters without modifiers, make SIMULATION == YES

```
#ifndef SIMULATION
  |
  || ((SIMULATION != NO) && (SIMULATION != YES))
#define SIMULATION      YES // Default: Either YES or NO
#endif
```
Define this to run the function that checks the compatibility between the chosen big number math library and the TPM code. Not all ports use this.

```c
#include ! (defined LIBRARY_COMPATABILITY_CHECK) || ((LIBRARY_COMPATABILITY_CHECK != NO) && (LIBRARY_COMPATABILITY_CHECK != YES))

#define LIBRARY_COMPATABILITY_CHECK YES // Default: Either YES or NO
#endif
```

Definition to allow alternate behavior for non-orderly startup. If there is a chance that the TPM could not update failedTries

```c
#include ! (defined USE_DA_USED) || (USE_DA_USED != NO) && (USE_DA_USED != YES))

#define USE_DA_USED YES // Default: Either YES or NO
#endif
```

Define TABLE_DRIVEN_DISPATCH to use tables rather than case statements for command dispatch and handle unmarshaling

```c
#include ! (defined TABLE_DRIVEN_DISPATCH) || ((TABLE_DRIVEN_DISPATCH != NO) && (TABLE_DRIVEN_DISPATCH != YES))

#define TABLE_DRIVEN_DISPATCH YES // Default: Either YES or NO
#endif
```

This switch is used to enable the self-test capability in AlgorithmTests.c

```c
#include ! (defined SELF_TEST) || ((SELF_TEST != NO) && (SELF_TEST != YES))

#define SELF_TEST YES // Default: Either YES or NO
#endif
```

Enable the generation of RSA primes using a sieve.

```c
#include ! (defined RSA_KEY_SIEVE) || (RSA_KEY_SIEVE != NO) && (RSA_KEY_SIEVE != YES))

#define RSA_KEY_SIEVE YES // Default: Either YES or NO
#endif
```

Enable the instrumentation of the sieve process. This is used to tune the sieve variables.

```c
#include RSA_KEY_SIEVE && SIMULATION

#if ! (defined RSA_INSTRUMENT) || (RSA_INSTRUMENT != NO) && (RSA_INSTRUMENT != YES))

#undef RSA_INSTRUMENT

#define RSA_INSTRUMENT NO // Default: Either YES or NO
#endif
```

This switch enables the RNG state save and restore

```c
#include !_DRBG_STATE_SAVE || (_DRBG_STATE_SAVE != NO) && (_DRBG_STATE_SAVE != YES))

#undef _DRBG_STATE_SAVE

#define _DRBG_STATE_SAVE
```

Switch added to support packed lists that leave out space associated with unimplemented commands. Comment this out to use linear lists.

NOTE: if vendor specific commands are present, the associated list is always in compressed form.

This switch indicates where clock epoch value should be stored. If this value defined, then it is assumed that the timer will change at any time so the nonce should be a random number kept in RAM. When it is not defined, then the timer only stops during power outages.

This switch allows use of #defines in place of pass-through marshaling or unmarshaling code. A pass-through function just calls another function to do the required function and does no parameter checking of its own. The table-driven dispatcher calls directly to the lowest level marshaling/unmarshaling code and by-passes any pass-through functions.

The switches in this group can only be enabled when doing debug during simulation

Enables use of the key cache. Default is YES

Enables use of a file to store the key cache values so that the TPM will start faster during debug. Default for this is YES
This provides fixed seeding of the RNG when doing debug on a simulator. This should allow consistent results on test runs as long as the input parameters to the functions remains the same. There is no default value.

```
#   if !(defined USE_DEBUG_RNG) || ((USE_DEBUG_RNG != NO) && (USE_DEBUG_RNG != YES))
#       undef USE_DEBUG_RNG
100 #   define USE_DEBUG_RNG YES // Default: Either YES or NO
101 #   endif
```

Don't change these. They are the settings needed when not doing a simulation and not doing debug. Can't use the key cache except during debug. Otherwise, all of the key values end up being the same

```
#   if DEBUG
```

In some cases, the relationship between two values may be dependent on things that change based on various selections like the chosen cryptographic libraries. It is possible that these selections will result in incompatible settings. These are often detectable by the compiler but it isn't always possible to do the check in the preprocessor code. For example, when the check requires use of 'sizeof()' then the preprocessor can't do the comparison. For these cases, we include a special macro that, depending on the compiler will generate a warning to indicate if the check always passes or always fails because it involves fixed constants. To run these checks, define COMPILER_CHECKS.

```
#   if !(defined COMPILER_CHECKS) || ((COMPILER_CHECKS != NO) && (COMPILER_CHECKS != YES))
#       undef COMPILER_CHECKS
111 #       define COMPILER_CHECKS NO // Default: Either YES or NO
112 #   endif
```

Some of the values (such as sizes) are the result of different options set in Implementation.h. The combination might not be consistent. A function is defined (TpmSizeChecks()) that is used to verify the sizes at run time. To enable the function, define this parameter.

```
#   if !(defined RUNTIME_SIZE_CHECKS) || ((RUNTIME_SIZE_CHECKS != NO) && (RUNTIME_SIZE_CHECKS != YES))
#       undef RUNTIME_SIZE_CHECKS
116 #       define RUNTIME_SIZE_CHECKS NO // Default: Either YES or NO
117 #   endif
```

If doing debug, can set the DRBG to print out the intermediate test values. Before enabling this, make sure that the dbgDumpMemBlock() function has been added someplace (preferably, somewhere in CryptRand.c)

```
#   if !(defined DRBG_DEBUG_PRINT) || ((DRBG_DEBUG_PRINT != NO) && (DRBG_DEBUG_PRINT != YES))
#       undef DRBG_DEBUG_PRINT
121 #       define DRBG_DEBUG_PRINT NO // Default: Either YES or NO
122 #   endif
```

If an assertion event it not going to produce any trace information (function and line number) then make FAIL_TRACE == NO

```
#   if !(defined FAIL_TRACE) || ((FAIL_TRACE != NO) && (FAIL_TRACE != YES))
#       undef FAIL_TRACE
125 #       define FAIL_TRACE YES // Default: Either YES or NO
126 #   endif
```
#ifdef // DEBUG

Indicate if the implementation is going to give lockout time credit for time up to the last orderly shutdown.

#define ACCUMULATE_SELF_HEAL_TIMER
|| ((ACCUMULATE_SELF_HEAL_TIMER != NO) && (ACCUMULATE_SELF_HEAL_TIMER != YES))
# undef ACCUMULATE_SELF_HEAL_TIMER
#define ACCUMULATE_SELF_HEAL_TIMER YES // Default: Either YES or NO
#endif

Indicates if the implementation is to compute the sizes of the proof and primary seed size values based on the implemented algorithms.

#if !(defined USE_SPEC_COMPLIANT_PROOFS)
|| ((USE_SPEC_COMPLIANT_PROOFS != NO) && (USE_SPEC_COMPLIANT_PROOFS != YES))
# undef USE_SPEC_COMPLIANT_PROOFS
#define USE_SPEC_COMPLIANT_PROOFS YES // Default: Either YES or NO
#endif

Comment this out to allow compile to continue even though the chosen proof values do not match the compliant values. This is written so that someone would have to proactively ignore errors.

#if !(defined SKIP_PROOF_ERRORS)
|| ((SKIP_PROOF_ERRORS != NO) && (SKIP_PROOF_ERRORS != YES))
# undef SKIP_PROOF_ERRORS
#define SKIP_PROOF_ERRORS NO // Default: Either YES or NO
#endif

This define is used to eliminate the use of bit-fields. It can be enabled for big- or little-endian machines. For big-endian architectures that numbers bits in registers from left to right (MSb0()) this must be enabled. Little-endian machines number from right to left with the least significant bit having assigned a bit number of 0. These are LSB0() machines they are also little-endian so they are also least-significant byte 0 (LSB0) machines. Big-endian (MSB0) machines may number in either direction (MSb0() or LSb0()). For an MSB0+MSb0() machine this value is required to be NO

#if !(defined USE_BIT_FIELD_STRUCTURES)
|| ((USE_BIT_FIELD_STRUCTURES != NO) && (USE_BIT_FIELD_STRUCTURES != YES))
# undef USE_BIT_FIELD_STRUCTURES
#define USE_BIT_FIELD_STRUCTURES YES // Default: Either YES or NO
#endif

This define is used to enable any runtime checks of the interface between the cryptographic library (e.g., OpenSSL()) and the thunking layer.

#if !(defined LIBRARY_COMPATIBILITY_CHECK)
|| ((LIBRARY_COMPATIBILITY_CHECK != NO) && (LIBRARY_COMPATIBILITY_CHECK != YES))
# undef LIBRARY_COMPATIBILITY_CHECK
#define LIBRARY_COMPATIBILITY_CHECK NO // Default: Either YES or NO
#endif

Change these definitions to turn all algorithms or commands ON or OFF. That is, to turn all algorithms on, set ALG_NO to YES. This is mostly useful as a debug feature.

#define ALG_YES      YES
#define ALG_NO       NO
#define CC_YES       YES
#define CC_NO        NO
#endif // _TPM_BUILD_SWITCHES_H_
5.20 TpmError.h

```c
#ifndef _TPM_ERROR_H
#define _TPM_ERROR_H
#define FATAL_ERROR_ALLOCATION              (1)
#define FATAL_ERROR_DIVIDE_ZERO             (2)
#define FATAL_ERROR_INTERNAL                (3)
#define FATAL_ERROR_PARAMETER               (4)
#define FATAL_ERROR_ENTROPY                 (5)
#define FATAL_ERROR_SELF_TEST               (6)
#define FATAL_ERROR_CRYPTO                  (7)
#define FATAL_ERROR_NV_UNRECOVERABLE        (8) // indicates that the TPM has
#define FATAL_ERROR_REMANUFACTURED         (9) // been re-manufactured after an
                                              // unrecoverable NV error
#define FATAL_ERROR_DRBG                    (10)
#define FATAL_ERROR_MOVE_SIZE               (11)
#define FATAL_ERROR_COUNTER_OVERFLOW        (12)
#define FATAL_ERROR_SUBTRACT                (13)
#define FATAL_ERROR_FORCED                  (16)
#define FATAL_ERROR_FORCED                  (66)
#endif // _TPM_ERROR_H
```
5.21 TpmTypes.h

```c
#ifndef _TPM_TYPES_H_
#define _TPM_TYPES_H_

typedef UINT32 TPM_ALGORITHM_ID;
typedef UINT32 TPM_MODIFIER_INDICATOR;
typedef UINT32 TPM_AUTHORIZATION_SIZE;
typedef UINT32 TPM_PARAMETER_SIZE;
typedef UINT16 TPM_KEY_SIZE;
typedef UINT16 TPM_KEY_BITS;

Table 2:5 - Definition of Types for Documentation Clarity

typedef UINT32 TPM_SPEC;
#define SPEC_FAMILY 0x322E3000
#define TPM_SPEC_FAMILY (TPM_SPEC)(SPEC_FAMILY)
#define SPEC_LEVEL 00
#define TPM_SPEC_LEVEL (TPM_SPEC)(SPEC_LEVEL)
#define SPEC_VERSION 150
#define TPM_SPEC_VERSION (TPM_SPEC)(SPEC_VERSION)
#define SPEC_YEAR 2018
#define TPM_SPEC_YEAR (TPM_SPEC)(SPEC_YEAR)
#define SPEC_DAY_OF_YEAR 262
#define TPM_SPEC_DAY_OF_YEAR (TPM_SPEC)(SPEC_DAY_OF_YEAR)

Table 2:6 - Definition of TPM_SPEC Constants

typedef UINT32 TPM_GENERATED;
#define TPM_GENERATED_VALUE (TPM_GENERATED)(0xFF544347)

Table 2:16 - Definition of TPM_RC Constants

typedef UINT32 TPM_RC;
#define TPM_RC_SUCCESS (TPM_RC)(0x000)
#define TPM_RC_BAD_TAG (TPM_RC)(0x01E)
#define TPM_RC_INITIALIZE (TPM_RC)(RC_VER1+0x000)
#define TPM_RC_FAILURE (TPM_RC)(RC_VER1+0x001)
#define TPM_RC_SEQUENCE (TPM_RC)(RC_VER1+0x003)
#define TPM_RC_PRIVATE (TPM_RC)(RC_VER1+0x00B)
#define TPM_RC_HMAC (TPM_RC)(RC_VER1+0x019)
#define TPM_RC_DISABLED (TPM_RC)(RC_VER1+0x020)
#define TPM_RC_EXCLUSIVE (TPM_RC)(RC_VER1+0x021)
#define TPM_RC_POLICY (TPM_RC)(RC_VER1+0x026)
#define TPM_RC_PCR (TPM_RC)(RC_VER1+0x027)
#define TPM_RC_PCR_CHANGED (TPM_RC)(RC_VER1+0x028)
#define TPM_RC_UPGRADE (TPM_RC)(RC_VER1+0x02D)
#define TPM_RC_TOO_MANY_CONTEXTS (TPM_RC)(RC_VER1+0x02E)
#define TPM_RC_AUTH_UNAVAILABLE (TPM_RC)(RC_VER1+0x02F)
#define TPM_RC_REBOOT (TPM_RC)(RC_VER1+0x030)
#define TPM_RC_UNBALANCED (TPM_RC)(RC_VER1+0x031)
#define TPM_RC_COMMAND_SIZE (TPM_RC)(RC_VER1+0x042)
#define TPM_RC_COMMAND_CODE (TPM_RC)(RC_VER1+0x043)
#define TPM_RC_AUTH_SIZE (TPM_RC)(RC_VER1+0x044)
#define TPM_RC_AUTH_CONTEXT (TPM_RC)(RC_VER1+0x045)
#define TPM_RC_NV_RANGE (TPM_RC)(RC_VER1+0x046)
#define TPM_RC_NV_SIZE (TPM_RC)(RC_VER1+0x047)
#define TPM_RC_NV_LOCKED (TPM_RC)(RC_VER1+0x048)
```
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>#define TPM_RC_NV_AUTHORIZATION (TPM_RC)(RC_VER1+0x049)</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>#define TPM_RC_NV_UNINITIALIZED (TPM_RC)(RC_VER1+0x04A)</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>#define TPM_RC_NV_SPACE (TPM_RC)(RC_VER1+0x04B)</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>#define TPM_RC_NV_DEFINED (TPM_RC)(RC_VER1+0x04C)</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>#define TPM_RC_BAD_CONTEXT (TPM_RC)(RC_VER1+0x050)</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>#define TPM_RC_CPHASH (TPM_RC)(RC_VER1+0x051)</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>#define TPM_RC_PARENT (TPM_RC)(RC_VER1+0x052)</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>#define TPM_RC_NEEDS_TEST (TPM_RC)(RC_VER1+0x053)</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>#define TPM_RC_NO_RESULT (TPM_RC)(RC_VER1+0x054)</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>#define TPM_RC_SENSITIVE (TPM_RC)(RC_VER1+0x055)</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>#define RC_MAX_FM0 (TPM_RC)(RC_VER1+0x07F)</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>#define RC_FMT1 (TPM_RC)(0x080)</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>#define TPM_RC_ASymmetric (TPM_RC)(RC_FMT1+0x001)</td>
<td></td>
</tr>
<tr>
<td>63</td>
<td>#define TPM_RC_ASymmetric (TPM_RC)(RC_FMT1+0x003)</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>#define TPM_RC_ATTRIBUTES (TPM_RC)(RC_FMT1+0x002)</td>
<td></td>
</tr>
<tr>
<td>65</td>
<td>#define TPM_RC_ATTRIBUTES (TPM_RC)(RC_FMT1+0x002)</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>#define TPM_RC_HASH (TPM_RC)(RC_FMT1+0x003)</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>#define TPM_RC_HASH (TPM_RC)(RC_FMT1+0x003)</td>
<td></td>
</tr>
<tr>
<td>68</td>
<td>#define TPM_RC_VALUE (TPM_RC)(RC_FMT1+0x004)</td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>#define TPM_RC_VALUE (TPM_RC)(RC_FMT1+0x004)</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>#define TPM_RC_HIERARCHY (TPM_RC)(RC_FMT1+0x005)</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>#define TPM_RC_HIERARCHY (TPM_RC)(RC_FMT1+0x005)</td>
<td></td>
</tr>
<tr>
<td>72</td>
<td>#define TPM_RC_KEY_SIZE (TPM_RC)(RC_FMT1+0x007)</td>
<td></td>
</tr>
<tr>
<td>73</td>
<td>#define TPM_RC_KEY_SIZE (TPM_RC)(RC_FMT1+0x007)</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>#define TPM_RC_MGF (TPM_RC)(RC_FMT1+0x008)</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>#define TPM_RC_MGF (TPM_RC)(RC_FMT1+0x008)</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>#define TPM_RC_MODE (TPM_RC)(RC_FMT1+0x009)</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>#define TPM_RC_MODE (TPM_RC)(RC_FMT1+0x009)</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>#define TPM_RC_TYPE (TPM_RC)(RC_FMT1+0x00A)</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>#define TPM_RC_TYPE (TPM_RC)(RC_FMT1+0x00A)</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>#define TPM_RC_HANDLE (TPM_RC)(RC_FMT1+0x00B)</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>#define TPM_RC_HANDLE (TPM_RC)(RC_FMT1+0x00B)</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>#define TPM_RC_KDF (TPM_RC)(RC_FMT1+0x00C)</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>#define TPM_RC_KDF (TPM_RC)(RC_FMT1+0x00C)</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>#define TPM_RC_RANGE (TPM_RC)(RC_FMT1+0x00D)</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>#define TPM_RC_RANGE (TPM_RC)(RC_FMT1+0x00D)</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>#define TPM_RC_AUTH_FAIL (TPM_RC)(RC_FMT1+0x00E)</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>#define TPM_RC_AUTH_FAIL (TPM_RC)(RC_FMT1+0x00E)</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>#define TPM_RC_NONCE (TPM_RC)(RC_FMT1+0x00F)</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>#define TPM_RC_NONCE (TPM_RC)(RC_FMT1+0x00F)</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>#define TPM_RC_PP (TPM_RC)(RC_FMT1+0x010)</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>#define TPM_RC_PP (TPM_RC)(RC_FMT1+0x010)</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>#define TPM_RC_SCHEME (TPM_RC)(RC_FMT1+0x012)</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>#define TPM_RC_SCHEME (TPM_RC)(RC_FMT1+0x012)</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>#define TPM_RC_SIZE (TPM_RC)(RC_FMT1+0x015)</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>#define TPM_RC_SIZE (TPM_RC)(RC_FMT1+0x015)</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>#define TPM_RC_SYMMETRIC (TPM_RC)(RC_FMT1+0x016)</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>#define TPM_RC_SYMMETRIC (TPM_RC)(RC_FMT1+0x016)</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>#define TPM_RC_TAG (TPM_RC)(RC_FMT1+0x017)</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>#define TPM_RC_TAG (TPM_RC)(RC_FMT1+0x017)</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>#define TPM_RC_SELECTOR (TPM_RC)(RC_FMT1+0x018)</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>#define TPM_RC_SELECTOR (TPM_RC)(RC_FMT1+0x018)</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>#define TPM_RC_INSUFFICIENT (TPM_RC)(RC_FMT1+0x01A)</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>#define TPM_RC_INSUFFICIENT (TPM_RC)(RC_FMT1+0x01A)</td>
<td></td>
</tr>
<tr>
<td>104</td>
<td>#define TPM_RC_SIGNATURE (TPM_RC)(RC_FMT1+0x01B)</td>
<td></td>
</tr>
<tr>
<td>105</td>
<td>#define TPM_RC_SIGNATURE (TPM_RC)(RC_FMT1+0x01B)</td>
<td></td>
</tr>
<tr>
<td>106</td>
<td>#define TPM_RC_KEY (TPM_RC)(RC_FMT1+0x01C)</td>
<td></td>
</tr>
<tr>
<td>107</td>
<td>#define TPM_RC_KEY (TPM_RC)(RC_FMT1+0x01C)</td>
<td></td>
</tr>
<tr>
<td>108</td>
<td>#define TPM_RC_POLICY_FAIL (TPM_RC)(RC_FMT1+0x01D)</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>#define TPM_RC_POLICY_FAIL (TPM_RC)(RC_FMT1+0x01D)</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>#define TPM_RC_INTEGRITY (TPM_RC)(RC_FMT1+0x01F)</td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>#define TPM_RC_INTEGRITY (TPM_RC)(RC_FMT1+0x01F)</td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>#define TPM_RC_TICKET (TPM_RC)(RC_FMT1+0x020)</td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>#define TPM_RC_TICKET (TPM_RC)(RC_FMT1+0x020)</td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>#define TPM_RC_RESERVED_BITS (TPM_RC)(RC_FMT1+0x021)</td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>#define TPM_RC_RESERVED_BITS (TPM_RC)(RC_FMT1+0x021)</td>
<td></td>
</tr>
</tbody>
</table>
#define TPM_RC_BAD_AUTH (TPM_RC)(RC_FMT1+0x022)
#define TPM_RCS_BAD_AUTH (TPM_RC)(RC_FMT1+0x022)
#define TPM_RC_EXPIRED (TPM_RC)(RC_FMT1+0x023)
#define TPM_RCS_EXPIRED (TPM_RC)(RC_FMT1+0x023)
#define TPM_RC_POLICY_CC (TPM_RC)(RC_FMT1+0x024)
#define TPM_RCS_POLICY_CC (TPM_RC)(RC_FMT1+0x024)
#define TPM_RC_BINDING (TPM_RC)(RC_FMT1+0x025)
#define TPM_RCS_BINDING (TPM_RC)(RC_FMT1+0x025)
#define TPM_RC_CURVE (TPM_RC)(RC_FMT1+0x026)
#define TPM_RCS_CURVE (TPM_RC)(RC_FMT1+0x026)
#define TPM_RC_ECC_POINT (TPM_RC)(RC_FMT1+0x027)
#define TPM_RCS_ECC_POINT (TPM_RC)(RC_FMT1+0x027)
#define RC_WARN (TPM_RC)(0x900)
#define TPM_RC_CONTEXT_GAP (TPM_RC)(RC_WARN+0x001)
#define TPM_RC_OBJECT_MEMORY (TPM_RC)(RC_WARN+0x002)
#define TPM_RC_SESSION_MEMORY (TPM_RC)(RC_WARN+0x003)
#define TPM_RC_MEMORY (TPM_RC)(RC_WARN+0x004)
#define TPM_RC_SESSION_HANDLES (TPM_RC)(RC_WARN+0x005)
#define TPM_RC_OBJECT_HANDLES (TPM_RC)(RC_WARN+0x006)
#define TPM_RC_LOCALITY (TPM_RC)(RC_WARN+0x007)
#define TPM_RC_YIELDED (TPM_RC)(RC_WARN+0x008)
#define TPM_RC_CANCELED (TPM_RC)(RC_WARN+0x009)
#define TPM_RC_TESTING (TPM_RC)(RC_WARN+0x00A)
#define TPM_RC_REFERENCE_H0 (TPM_RC)(RC_WARN+0x010)
#define TPM_RC_REFERENCE_H1 (TPM_RC)(RC_WARN+0x011)
#define TPM_RC_REFERENCE_H2 (TPM_RC)(RC_WARN+0x012)
#define TPM_RC_REFERENCE_H3 (TPM_RC)(RC_WARN+0x013)
#define TPM_RC_REFERENCE_H4 (TPM_RC)(RC_WARN+0x014)
#define TPM_RC_REFERENCE_H5 (TPM_RC)(RC_WARN+0x015)
#define TPM_RC_REFERENCE_H6 (TPM_RC)(RC_WARN+0x016)
#define TPM_RC_REFERENCE_S0 (TPM_RC)(RC_WARN+0x018)
#define TPM_RC_REFERENCE_S1 (TPM_RC)(RC_WARN+0x019)
#define TPM_RC_REFERENCE_S2 (TPM_RC)(RC_WARN+0x01A)
#define TPM_RC_REFERENCE_S3 (TPM_RC)(RC_WARN+0x01B)
#define TPM_RC_REFERENCE_S4 (TPM_RC)(RC_WARN+0x01C)
#define TPM_RC_REFERENCE_S5 (TPM_RC)(RC_WARN+0x01D)
#define TPM_RC_REFERENCE_S6 (TPM_RC)(RC_WARN+0x01E)
#define TPM_RC_NV_RATE (TPM_RC)(RC_WARN+0x020)
#define TPM_RC_LOCKOUT (TPM_RC)(RC_WARN+0x021)
#define TPM_RC_RETRY (TPM_RC)(RC_WARN+0x022)
#define TPM_RC_NV_UNAVAILABLE (TPM_RC)(RC_WARN+0x023)
#define TPM_RC_NOT_USED (TPM_RC)(RC_WARN+0x07F)
#define TPM_RC_H (TPM_RC)(0x000)
#define TPM_RC_P (TPM_RC)(0x040)
#define TPM_RC_S (TPM_RC)(0x080)
#define TPM_RC_1 (TPM_RC)(0x0C0)
#define TPM_RC_2 (TPM_RC)(0x100)
#define TPM_RC_3 (TPM_RC)(0x140)
#define TPM_RC_4 (TPM_RC)(0x180)
#define TPM_RC_5 (TPM_RC)(0x1C0)
#define TPM_RC_6 (TPM_RC)(0x200)
#define TPM_RC_7 (TPM_RC)(0x240)
#define TPM_RC_8 (TPM_RC)(0x280)
#define TPM_RC_9 (TPM_RC)(0x2C0)
#define TPM_RC_A (TPM_RC)(0x300)
#define TPM_RC_B (TPM_RC)(0x340)
#define TPM_RC_C (TPM_RC)(0x380)
#define TPM_RC_D (TPM_RC)(0x3C0)
#define TPM_RC_E (TPM_RC)(0x400)
#define TPM_RC_F (TPM_RC)(0x440)
#define TPM_RC_N_MASK (TPM_RC)(RC_WARN+0xF00)

Table 2:17 - Definition of TPM_CLOCK_ADJUST Constants

typedef INT8 TPM_CLOCK_ADJUST;
#define TPM_CLOCK_COARSE_SLOWER (TPM_CLOCK_ADJUST)(3)
#define TPM_CLOCK_MEDIUM_SLOWER (TPM_CLOCK_ADJUST)(2)
#define TPM_CLOCK_FINE_SLOWER (TPM_CLOCK_ADJUST)(1)
#define TPM_CLOCK_NO_CHANGE (TPM_CLOCK_ADJUST)(0)
#define TPM_CLOCK_FINE_FASTER (TPM_CLOCK_ADJUST)(1)
#define TPM_CLOCK_MEDIUM_FASTER (TPM_CLOCK_ADJUST)(2)
#define TPM_CLOCK_COARSE_FASTER (TPM_CLOCK_ADJUST)(3)

Table 2:18 - Definition of TPM_EO Constants

typedef UINT16 TPM_EO;
define TPM_EO_EQ (TPM_EO)(0x0000)
define TPM_EO_NEQ (TPM_EO)(0x0001)
define TPM_EO_SIGNED_GT (TPM_EO)(0x0002)
define TPM_EO_UNSIGNED_GT (TPM_EO)(0x0003)
define TPM_EO_SIGNED_LT (TPM_EO)(0x0004)
define TPM_EO_UNSIGNED_LT (TPM_EO)(0x0005)
define TPM_EO_SIGNED_GE (TPM_EO)(0x0006)
define TPM_EO_UNSIGNED_GE (TPM_EO)(0x0007)
define TPM_EO_SIGNED_LE (TPM_EO)(0x0008)
define TPM_EO_UNSIGNED_LE (TPM_EO)(0x0009)
define TPM_EO_BITSET (TPM_EO)(0x000A)
define TPM_EO_BITCLEAR (TPM_EO)(0x000B)

Table 2:19 - Definition of TPM_ST Constants

typedef UINT16 TPM_ST;
define TPM_ST_RSP_COMMAND (TPM_ST)(0x00C4)
define TPM_ST_NULL (TPM_ST)(0x8000)
define TPM_ST_NO_SESSIONS (TPM_ST)(0x8001)
define TPM_ST_SESSIONS (TPM_ST)(0x8002)
define TPM_ST_ATTEST_NV (TPM_ST)(0x8014)
define TPM_ST_ATTEST_COMMAND_AUDIT (TPM_ST)(0x8015)
define TPM_ST_ATTEST_SESSION_AUDIT (TPM_ST)(0x8016)
define TPM_ST_ATTEST_CERTIFY (TPM_ST)(0x8017)
define TPM_ST_ATTEST_QUOTE (TPM_ST)(0x8018)
define TPM_ST_ATTEST_TIME (TPM_ST)(0x8019)
define TPM_ST_ATTEST_CREATION (TPM_ST)(0x801A)
define TPM_ST_CREATION (TPM_ST)(0x8021)
define TPM_ST_VERIFIED (TPM_ST)(0x8022)
define TPM_ST_AUTH_SECRET (TPM_ST)(0x8023)
define TPM_ST_HASHCHECK (TPM_ST)(0x8024)
define TPM_ST_AUTH_SIGNED (TPM_ST)(0x8025)
define TPM_ST_CERTIFICATE (TPM_ST)(0x8029)

Table 2:20 - Definition of TPM_SU Constants

typedef UINT16 TPM_SU;
define TPM_SUCLEAR (TPM_SU)(0x0000)
define TPM_SU_STATE (TPM_SU)(0x0001)

Table 2:21 - Definition of TPM_SE Constants

typedef UINT8 TPM_SE;
define TPM_SE_HMAC (TPM_SE)(0x00)
define TPM_SE_POLICY (TPM_SE)(0x01)
define TPM_SE_TRIAL (TPM_SE)(0x03)

Table 2:22 - Definition of TPM_CAP Constants

typedef UINT32 TPM_CAP;
define TPM_CAP_FIRST (TPM_CAP)(0x00000000)
define TPM_CAP_ALGS (TPM_CAP)(0x00000000)
#define TPM_CAP_HANDLES    (TPM_CAP)(0x00000001)
#define TPM_CAP_COMMANDS   (TPM_CAP)(0x00000002)
#define TPM_CAP_PP_COMMANDS (TPM_CAP)(0x00000003)
#define TPM_CAP_AUDIT_COMMANDS (TPM_CAP)(0x00000004)
#define TPM_CAP_PCRS        (TPM_CAP)(0x00000005)
#define TPM_CAP_TP_PROPERTIES (TPM_CAP)(0x00000006)
#define TPM_CAP_PCR_PROPERTIES (TPM_CAP)(0x00000007)
#define TPM_CAP_ECC_CURVES  (TPM_CAP)(0x00000008)
#define TPM_CAP_AUTH_POLICIES (TPM_CAP)(0x00000009)
#define TPM_CAP_LAST        (TPM_CAP)(0x00000009)
#define TPM_CAP_VENDOR_PROPERTY (TPM_CAP)(0x00000100)

Table 2:23 - Definition of TPM_PT Constants

typedef UINT32 TPM_PT;

#define TPM_PT_NONE          (TPM_PT)(0x00000000)
#define PT_GROUP             (TPM_PT)(0x00000001)
#define TPM_PT_FAMILY_INDICATOR (TPM_PT)(PT_GROUP*1)
#define TPM_PT_LEVEL          (TPM_PT)(PT_FIXED+0)
#define TPM_PT_REVISION       (TPM_PT)(PT_FIXED+1)
#define TPM_PT_DAY_OF_YEAR    (TPM_PT)(PT_FIXED+2)
#define TPM_PT_YEAR           (TPM_PT)(PT_FIXED+3)
#define TPM_PT_MANUFACTURER   (TPM_PT)(PT_FIXED+4)
#define TPM_PT_VENDOR_STRING_1 (TPM_PT)(PT_FIXED+5)
#define TPM_PT_VENDOR_STRING_2 (TPM_PT)(PT_FIXED+6)
#define TPM_PT_VENDOR_STRING_3 (TPM_PT)(PT_FIXED+7)
#define TPM_PT_VENDOR_STRING_4 (TPM_PT)(PT_FIXED+8)
#define TPM_PT_VENDOR_TPM_TYPE (TPM_PT)(PT_FIXED+9)
#define TPM_PT_FIRMWARE_VERSION_1 (TPM_PT)(PT_FIXED+10)
#define TPM_PT_FIRMWARE_VERSION_2 (TPM_PT)(PT_FIXED+11)
#define TPM_PT_INPUT_BUFFER   (TPM_PT)(PT_FIXED+12)
#define TPM_PT_HR_TRANSIENT_MIN (TPM_PT)(PT_FIXED+13)
#define TPM_PT_HR_PERSISTENT_MIN (TPM_PT)(PT_FIXED+14)
#define TPM_PT_HR_LOADED_MIN   (TPM_PT)(PT_FIXED+15)
#define TPM_PT_ACTIVE_SESSIONS_MAX (TPM_PT)(PT_FIXED+16)
#define TPM_PT_PCR_COUNT      (TPM_PT)(PT_FIXED+17)
#define TPM_PT_PCR_SELECT_MIN (TPM_PT)(PT_FIXED+18)
#define TPM_PT_CONTEXT_GAP_MAX (TPM_PT)(PT_FIXED+19)
#define TPM_PT_NV_COUNTERS_MAX (TPM_PT)(PT_FIXED+20)
#define TPM_PT_NV_INDEX_MAX   (TPM_PT)(PT_FIXED+21)
#define TPM_PT_MEMORY         (TPM_PT)(PT_FIXED+22)
#define TPM_PT_CLOCK_UPDATE   (TPM_PT)(PT_FIXED+23)
#define TPM_PT_CONTEXT_HASH   (TPM_PT)(PT_FIXED+24)
#define TPM_PT_CONTEXT_SYM    (TPM_PT)(PT_FIXED+25)
#define TPM_PT_CONTEXT_SYM_SIZE (TPM_PT)(PT_FIXED+26)
#define TPM_PT_ORDERLY_COUNT (TPM_PT)(PT_FIXED+27)
#define TPM_PT_MAX_COMMAND_SIZE (TPM_PT)(PT_FIXED+28)
#define TPM_PT_MAX_RESPONSE_SIZE (TPM_PT)(PT_FIXED+29)
#define TPM_PT_MAX_DIGEST    (TPM_PT)(PT_FIXED+30)
#define TPM_PT_MAX_OBJECT_CONTEXT (TPM_PT)(PT_FIXED+31)
#define TPM_PT_MAX_SESSION_CONTEXT (TPM_PT)(PT_FIXED+32)
#define TPM_PT_PS_FAMILY_INDICATOR (TPM_PT)(PT_FIXED+33)
#define TPM_PT_PS_LEVEL       (TPM_PT)(PT_FIXED+34)
#define TPM_PT_PS_REVISION    (TPM_PT)(PT_FIXED+35)
#define TPM_PT_PS_DAY_OF_YEAR (TPM_PT)(PT_FIXED+36)
#define TPM_PT_PS_YEAR        (TPM_PT)(PT_FIXED+37)
#define TPM_PT_SPLIT_MAX      (TPM_PT)(PT_FIXED+38)
#define TPM_PT_TOTAL_COMMANDS (TPM_PT)(PT_FIXED+39)
#define TPM_PT_LIBRARY_COMMANDS (TPM_PT)(PT_FIXED+40)
#define TPM_PT_VENDOR_COMMANDS (TPM_PT)(PT_FIXED+41)
#define TPM_PT_NV_BUFFER_MAX  (TPM_PT)(PT_FIXED+42)
#define TPM_PT_MODES          (TPM_PT)(PT_FIXED+43)
#define TPM_PT_MAX_CAP_BUFFER (TPM_PT)(PT_FIXED+44)
#define TPM_PT_MAX_CAP_BUFFER (TPM_PT)(PT_FIXED+45)
#define TPM_PT_MAX_CAP_BUFFER (TPM_PT)(PT_FIXED+46)
#define PT_VAR                (TPM_PT)(PT_GROUP*2)
#define TPM_PT_PERMANENT (TPM_PT) (PT_VAR+0)
#define TPM_PT_STARTUP_CLEAR (TPM_PT) (PT_VAR+1)
#define TPM_PT_MR_NV_INDEX (TPM_PT) (PT_VAR+2)
#define TPM_PT_MR_LOADED (TPM_PT) (PT_VAR+3)
#define TPM_PT_MR_LOADED_AVAIL (TPM_PT) (PT_VAR+4)
#define TPM_PT_MR_ACTIVE (TPM_PT) (PT_VAR+5)
#define TPM_PT_MR_ACTIVE_AVAIL (TPM_PT) (PT_VAR+6)
#define TPM_PT_MR_TRANSIENT_AVAIL (TPM_PT) (PT_VAR+7)
#define TPM_PT_MR_PERSISTENT (TPM_PT) (PT_VAR+8)
#define TPM_PT_MR_PERSISTENT_AVAIL (TPM_PT) (PT_VAR+9)
#define TPM_PT_NV_COUNTERS (TPM_PT) (PT_VAR+10)
#define TPM_PT_NV_COUNTERS_AVAIL (TPM_PT) (PT_VAR+11)
#define TPM_PT_ALGORITHM_SET (TPM_PT) (PT_VAR+12)
#define TPM_PT_LOADED_CURVES (TPM_PT) (PT_VAR+13)
#define TPM_PT_LOCKOUT_COUNTER (TPM_PT) (PT_VAR+14)
#define TPM_PT_MAX_AUTH_FAIL (TPM_PT) (PT_VAR+15)
#define TPM_PT_LOCKOUT_INTERVAL (TPM_PT) (PT_VAR+16)
#define TPM_PT_LOCKOUT_RECOVERY (TPM_PT) (PT_VAR+17)
#define TPM_PT_NV_WRITE_RECOVERY (TPM_PT) (PT_VAR+18)
#define TPM_PT_AUDIT_COUNTER_0 (TPM_PT) (PT_VAR+19)
#define TPM_PT_AUDIT_COUNTER_1 (TPM_PT) (PT_VAR+20)

Table 2:24 - Definition of TPM_PT_PCR Constants

typedef UINT32 TPM_PT_PCR;
#define TPM_PT_PCR_FIRST (TPM_PT_PCR) (0x00000000)
#define TPM_PT_PCR_SAVE (TPM_PT_PCR) (0x00000001)
#define TPM_PT_PCR_EXTEND_L0 (TPM_PT_PCR) (0x00000002)
#define TPM_PT_PCR_RESET_L0 (TPM_PT_PCR) (0x00000003)
#define TPM_PT_PCR_EXTEND_L1 (TPM_PT_PCR) (0x00000004)
#define TPM_PT_PCR_RESET_L1 (TPM_PT_PCR) (0x00000005)
#define TPM_PT_PCR_EXTEND_L2 (TPM_PT_PCR) (0x00000006)
#define TPM_PT_PCR_RESET_L2 (TPM_PT_PCR) (0x00000007)
#define TPM_PT_PCR_EXTEND_L3 (TPM_PT_PCR) (0x00000008)
#define TPM_PT_PCR_RESET_L3 (TPM_PT_PCR) (0x00000009)
#define TPM_PT_PCR_EXTEND_L4 (TPM_PT_PCR) (0x0000000A)
#define TPM_PT_PCR_RESET_L4 (TPM_PT_PCR) (0x0000000B)
#define TPM_PT_PCR_NO_INCREMENT (TPM_PT_PCR) (0x0000000C)
#define TPM_PT_PCR_DRTM_RESET (TPM_PT_PCR) (0x0000000D)
#define TPM_PT_PCR_POLICY (TPM_PT_PCR) (0x0000000E)
#define TPM_PT_PCR_AUTH (TPM_PT_PCR) (0x0000000F)

Table 2:25 - Definition of TPM_PS Constants

typedef UINT32 TPM_PS;
#define TPM_PS_MAIN (TPM_PS) (0x00000000)
#define TPM_PS_PC (TPM_PS) (0x00000001)
#define TPM_PS_PDA (TPM_PS) (0x00000002)
#define TPM_PS_CELL_PHONE (TPM_PS) (0x00000003)
#define TPM_PS_SERVER (TPM_PS) (0x00000004)
#define TPM_PS_PERIPHERAL (TPM_PS) (0x00000005)
#define TPM_PS_TSS (TPM_PS) (0x00000006)
#define TPM_PS_STORAGE (TPM_PS) (0x00000007)
#define TPM_PS_AUTHENTICATION (TPM_PS) (0x00000008)
#define TPM_PS_EMBEDDED (TPM_PS) (0x00000009)
#define TPM_PS_HARDCOPY (TPM_PS) (0x0000000A)
#define TPM_PS_INFRASTRUCTURE (TPM_PS) (0x0000000B)
#define TPM_PS_VIRTUALIZATION (TPM_PS) (0x0000000C)
#define TPM_PS_TNC (TPM_PS) (0x0000000D)
#define TPM_PS_MULTI_TENANT (TPM_PS) (0x0000000E)
#define TPM_PS_TC (TPM_PS) (0x0000000F)

Table 2:26 - Definition of Types for Handles
typedef UINT32 TPM_HANDLE;

Table 2:27 - Definition of TPM_HT Constants

typedef UINT8 TPM_HT;
#define TPM_HT_PCR (TPM_HT)(0x00)
#define TPM_HT_NV_INDEX (TPM_HT)(0x01)
#define TPM_HT_HMAC_SESSION (TPM_HT)(0x02)
#define TPM_HT_LOADED_SESSION (TPM_HT)(0x02)
#define TPM_HT_POLICY_SESSION (TPM_HT)(0x03)
#define TPM_HT_SAVED_SESSION (TPM_HT)(0x03)
#define TPM_HT_LOADED_SESSION (TPM_HT)(0x03)
#define TPM_HT_PERMANENT (TPM_HT)(0x40)
#define TPM_HT_TRANSIENT (TPM_HT)(0x80)
#define TPM_HT_PERSISTENT (TPM_HT)(0x81)
#define TPM_HT_AC (TPM_HT)(0x90)

Table 2:28 - Definition of TPM_RH Constants

typedef TPM_HANDLE TPM_RH;
#define TPM_RH_FIRST (TPM_RH)(0x40000000)
#define TPM_RH_SRK (TPM_RH)(0x40000000)
#define TPM_RH_OWNER (TPM_RH)(0x40000001)
#define TPM_RH_REVOKE (TPM_RH)(0x40000002)
#define TPM_RH_TRANSPORT (TPM_RH)(0x40000003)
#define TPM_RH_OPERATOR (TPM_RH)(0x40000004)
#define TPM_RH_ADMIN (TPM_RH)(0x40000005)
#define TPM_RH_EK (TPM_RH)(0x40000006)
#define TPM_RH_NULL (TPM_RH)(0x40000007)
#define TPM_RH_UNASSIGNED (TPM_RH)(0x40000008)
#define TPM_RS_PW (TPM_RH)(0x40000009)
#define TPM_RH_LOCKOUT (TPM_RH)(0x4000000A)
#define TPM_RH_ENDORSEMENT (TPM_RH)(0x4000000B)
#define TPM_RH_PLATFORM (TPM_RH)(0x4000000C)
#define TPM_RH_PLATFORM_NV (TPM_RH)(0x4000000D)
#define TPM_RH_AUTH_00 (TPM_RH)(0x40000010)
#define TPM_RH_AUTHF (TPM_RH)(0x40000010)
#define TPM_RH_LAST (TPM_RH)(0x4000001F)

Table 2:29 - Definition of TPM_HC Constants

typedef TPM_HANDLE TPM_HC;
#define HR_HANDLE_MASK (TPM_HC)(0x00FFFFFF)
#define HR_RANGE_MASK (TPM_HC)(0xFF000000)
#define HR_SHIFT (TPM_HC)(24)
#define HR_PCR (TPM_HC)((TPM_HT_PCR<<HR_SHIFT))
#define HR_HMAC_SESSION (TPM_HC)((TPM_HT_HMAC_SESSION<<HR_SHIFT))
#define HR_POLICY_SESSION (TPM_HC)((TPM_HT_POLICY_SESSION<<HR_SHIFT))
#define HR_TRANSIENT (TPM_HC)((TPM_HT_TRANSIENT<<HR_SHIFT))
#define HR_PERSISTENT (TPM_HC)((TPM_HT_PERSISTENT<<HR_SHIFT))
#define HR_NV_INDEX (TPM_HC)((TPM_HT_NV_INDEX<<HR_SHIFT))
#define PCR_FIRST (TPM_HC)((TPM_HC)(0x00000000))
#define PCR_LAST (TPM_HC)(((TPM_HC)(0x00000000)+IMPLEMENTATION_PCR-1))
#define HMAC_SESSION_FIRST (TPM_HC)(HR_HMAC_SESSION+0)
#define HMAC_SESSION_LAST (TPM_HC)((HMAC_SESSION_FIRST+MAX_ACTIVE_SESSIONS-1))
#define LOADED_SESSION_FIRST (TPM_HC)(HMAC_SESSION_FIRST)
#define LOADEDSESSION_LAST (TPM_HC)(HMAC_SESSION_LAST)
#define POLICY_SESSION_FIRST (TPM_HC)(HR_POLICY_SESSION+0)
#define POLICY_SESSION_LAST (TPM_HC)((POLICY_SESSION_FIRST+MAX_ACTIVE_SESSIONS-1))
#define TRANSIENT_FIRST (TPM_HC)(HR_TRANSIENT+0)
#define ACTIVE_SESSION_FIRST (TPM_HC)(POLICY_SESSION_FIRST)
#define ACTIVE_SESSION_LAST (TPM_HC)(POLICY_SESSION_LAST)
#define TRANSIENT_LAST (TPM_HC)((TRANSIENT_FIRST+MAX_LOADED_OBJECTS-1))
#define PERSISTENT_FIRST (TPM_HC)((HR_PERSISTENT+0))
#define PERSISTENT_LAST (TPM_HC)((PERSISTENT_FIRST+0x00FFFFFF))
#define PLATFORM_PERSISTENT (TPM_HC)((PERSISTENT_FIRST+0x00800000))
#define NV_INDEX_FIRST (TPM_HC)((HR_NV_INDEX+0))
#define NV_INDEX_LAST (TPM_HC)((NV_INDEX_FIRST+0x00FFFFFF))
#define PLATFORM_PERSISTENT (TPM_HC)((PERSISTENT_FIRST+0x00800000))
#define NV_INDEX_FIRST (TPM_HC)((HR_NV_INDEX+0))
#define NV_INDEX_LAST (TPM_HC)((HR_NV_INDEX+0x00FFFFFF))
#define PERMANENT_FIRST (TPM_HC)(TPM_RH_FIRST)
#define PERMANENT_LAST (TPM_HC)(TPM_RH_LAST)
#define HR_NV_AC (((TPM_HT_NV_INDEX<<HR_SHIFT)+0xD00000))
#define NV_AC_FIRST (TPM_HC)((HR_NV_AC+0))
#define NV_AC_LAST (TPM_HC)((HR_NV_AC+0x0000FFFF))
#define HR_AC (TPM_HC)((TPM_HT_AC<<HR_SHIFT))
#define AC_FIRST (TPM_HC)((HR_AC+0))
#define AC_LAST (TPM_HC)((HR_AC+0x0000FFFF))

#if USE_BIT_FIELD_STRUCTURES
typedef struct TPMA_ALGORITHM {
  // Table 2:30
  unsigned asymmetric : 1;
  unsigned symmetric : 1;
  unsigned hash : 1;
  unsigned object : 1;
  unsigned Reserved_bits_at_4 : 4;
  unsigned signing : 1;
  unsigned encrypting : 1;
  unsigned method : 1;
  unsigned Reserved_bits_at_11 : 21;
} TPMA_ALGORITHM;
/* Bits */
#endif

#define TPMA_ALGORITHM_INITIALIZER(asymmetric, symmetric, hash, object, bits_at_4, signing, encrypting, method, bits_at_11) {
  asymmetric, symmetric, hash, object, bits_at_4, signing, encrypting, method, bits_at_11
}

#if USE_BIT_FIELD_STRUCTURES
typedef struct TPMA_OBJECT {
  // Table 2:31
  unsigned Reserved_bit_at_0 : 1;
  unsigned fixedTPM : 1;
  unsigned stClear : 1;
  unsigned Reserved_bit_at_3 : 1;
  unsigned fixedParent : 1;
  unsigned sensitiveDataOrigin : 1;
  unsigned userWithAuth : 1;
}
#endif
unsigned adminWithPolicy : 1;
unsigned Reserved_bits_at_8 : 2;
unsigned noDA : 1;
unsigned encryptedDuplication : 1;
unsigned Reserved_bits_at_12 : 4;
unsigned restricted : 1;
unsigned decrypt : 1;
unsigned sign : 1;
unsigned Reserved_bits_at_19 : 13;
}
/* Bits */

This is the initializer for a TPMA_OBJECT structure

#define TPMA_OBJECT_INITIALIZER(
  bit_at_0,             fixedtpm,             stclear,
  bit_at_3,             fixedparent,          sensitivedataorigin,
  userwithauth,         adminwithpolicy,      bits_at_8,
  noda,                 encryptedduplication, bits_at_12,
  restricted,           decrypt,              sign,
  bits_at_19)

((fixedtpm << 1)              + (stclear << 2)               +
(fixedparent << 4)           + (sensitivedataorigin << 5)   +
(userwithauth << 6)          + (adminwithpolicy << 7)       +
(noda << 10)                 + (encryptedduplication << 11) +
(restricted << 16)           + (decrypt << 17))

#else // USE_BIT_FIELD_STRUCTURES

This implements Table 2:31 TPMA_OBJECT using bit masking

typedef UINT32                  TPMA_OBJECT;
#define TPMA_OBJECT_fixedTPM       ((TPMA_OBJECT)1 << 1)
#define TPMA_OBJECT_stClear        ((TPMA_OBJECT)1 << 2)
#define TPMA_OBJECT_fixedParent    ((TPMA_OBJECT)1 << 4)
#define TPMA_OBJECT_sensitiveDataOrigin ((TPMA_OBJECT)1 << 5)
#define TPMA_OBJECT_userWithAuth   ((TPMA_OBJECT)1 << 6)
#define TPMA_OBJECT_adminWithPolicy((TPMA_OBJECT)1 << 7)
#define TPMA_OBJECT_noDA           ((TPMA_OBJECT)1 << 10)
#define TPMA_OBJECT_encryptedDuplication ((TPMA_OBJECT)1 << 11)
#define TPMA_OBJECT_restricted     ((TPMA_OBJECT)1 << 16)
#define TPMA_OBJECT_decrypt        ((TPMA_OBJECT)1 << 17)
#define TPMA_OBJECT_sign           ((TPMA_OBJECT)1 << 18)

This is the initializer for a TPMA_OBJECT bit array.

#define TPMA_OBJECT_INITIALIZER(
  bit_at_0,             fixedtpm,             stclear,
  bit_at_3,             fixedparent,          sensitivedataorigin,
  userwithauth,         adminwithpolicy,      bits_at_8,
  noda,                 encryptedduplication, bits_at_12,
  restricted,           decrypt,              sign,
  bits_at_19)

((fixedtpm << 1)              + (stclear << 2)               +
(fixedparent << 4)           + (sensitivedataorigin << 5)   +
(userwithauth << 6)          + (adminwithpolicy << 7)       +
(noda << 10)                 + (encryptedduplication << 11) +
(restricted << 16)           + (decrypt << 17))

#define USE_BIT_FIELD_STRUCTURES

if USE_BIT_FIELD_STRUCTURES
typedef struct TPMA_SESSION {
  // Table 2:32
  unsigned continueSession : 1;
  unsigned auditExclusive : 1;
  unsigned auditReset : 1;
}
unsigned Reserved_bits_at_3 : 2;
unsigned decrypt : 1;
unsigned encrypt : 1;
unsigned audit : 1;
} TPMA_SESSION;

This is the initializer for a TPMA_SESSION structure

#define TPMA_SESSION_INITIALIZER(  
continuesession, auditexclusive, auditreset, bits_at_3,  
decrypt, encrypt, audit)  
{continuesession, auditexclusive, auditreset, bits_at_3,  
decrypt, encrypt, audit}
#endif

This implements Table 2:32 TPMA_SESSION using bit masking

typedef UINT8 TPMA_SESSION;
#define TPMA_SESSION_continueSession ((TPMA_SESSION)1 << 0)
#define TPMA_SESSION_auditExclusive ((TPMA_SESSION)1 << 1)
#define TPMA_SESSION_auditReset ((TPMA_SESSION)1 << 2)
#define TPMA_SESSION_decrypt ((TPMA_SESSION)1 << 5)
#define TPMA_SESSION_encrypt ((TPMA_SESSION)1 << 6)
#define TPMA_SESSION_audit ((TPMA_SESSION)1 << 7)

This is the initializer for a TPMA_SESSION bit array.

#define TPMA_SESSION_INITIALIZER(  
continuesession, auditexclusive, auditreset, bits_at_3,  
decrypt, encrypt, audit)  
((continuesession << 0) + (auditexclusive << 1) +  
(auditreset << 2) + (decrypt << 5) +  
(encrypt << 6) + (audit << 7))
#endif

This implements Table 2:33 TPMA_SESSION using bit masking

typedef struct TPMA_LOCALITY {  
TPM_LOC_ZERO : 1;
TPM_LOC_ONE: 1;
TPM_LOC_TWO: 1;
TPM_LOC_THREE: 1;
TPM_LOC_FOUR: 1;
Extended : 3;
} TPMA_LOCALITY;

This is the initializer for a TPMA_LOCALITY structure

#define TPMA_LOCALITY_INITIALIZER(  
tpm_loc_zero, tpm_loc_one, tpm_loc_two, tpm_loc_three,  
tpm_loc_four, extended)  
{tpm_loc_zero, tpm_loc_one, tpm_loc_two, tpm_loc_three,  
tpm_loc_four, extended}
#endif

This implements Table 2:33 TPMA_LOCALITY using bit masking

typedef UINT8 TPMA_LOCALITY;
#define TPMA_LOCALITY_TPM_LOC_ZERO ((TPMA_LOCALITY)1 << 0)
#define TPMA_LOCALITY_TPM_LOC_ONE ((TPMA_LOCALITY)1 << 1)
#define TPMA_LOCALITY_TPM_LOC_TWO ((TPMA_LOCALITY)1 << 2)
#define TPMA_LOCALITY_TPM_LOC_THREE ((TPMA_LOCALITY)1 << 3)
#define TPMA_LOCALITY_TPM_LOC_FOUR ((TPMA_LOCALITY)1 << 4)
#define TPMA_LOCALITY_Extended_SHIFT 5
#define TPMA_LOCALITY_Extended ((TPMA_LOCALITY)0x7 << 5)
This is the initializer for a TPMA_LOCALITY bit array.

```
#define TPMA_LOCALITY_INITIALIZER(
  tpm_loc_zero, tpm_loc_one, tpm_loc_two, tpm_loc_three,
  tpm_loc_four, extended) ((tpm_loc_zero << 0) + (tpm_loc_one << 1) + (tpm_loc_two << 2) +
  (tpm_loc_three << 3) + (tpm_loc_four << 4) + (extended << 5))
#endif // USE_BIT_FIELD_STRUCTURES
```

This is the initializer for a TPMA_PERMANENT structure

```
#define TPMA_PERMANENT_INITIALIZER(
  ownerauthset, endorsementauthset, lockoutauthset,
  bits_at_3, disableclear, inlockout,
  tpmgeneratedeps, bits_at_11)
```

This implements Table 2:34 TPMA_PERMANENT using bit masking

```
typedef UINT32 TPMA_PERMANENT;
#define TPMA_PERMANENT_ownerAuthSet ((TPMA_PERMANENT)1 << 0)
#define TPMA_PERMANENT_endorsementAuthSet ((TPMA_PERMANENT)1 << 1)
#define TPMA_PERMANENT_lockoutAuthSet ((TPMA_PERMANENT)1 << 2)
#define TPMA_PERMANENT_disableClear ((TPMA_PERMANENT)1 << 8)
#define TPMA_PERMANENT_inLockout ((TPMA_PERMANENT)1 << 9)
#define TPMA_PERMANENT_tpmGeneratedEPS ((TPMA_PERMANENT)1 << 10)
```

This is the initializer for a TPMA_PERMANENT bit array.

```
#define TPMA_PERMANENT_INITIALIZER(
  ownerauthset, endorsementauthset, lockoutauthset,
  bits_at_3, disableclear, inlockout,
  tpmgeneratedeps, bits_at_11)
```

This is the initializer for a TPMA_STARTUP_CLEAR structure

```
#define TPMA_STARTUP_CLEAR_INITIALIZER(
  phEnable, shEnable, ehEnable, phEnableNV,
  reserved_bits_at_4, orderly)
```

This implements Table 2:35 TPMA_STARTUP_CLEAR using bit masking

```
typedef struct TPMA_STARTUP_CLEAR {
  phEnable : 1;
  shEnable : 1;
  ehEnable : 1;
  phEnableNV : 1;
  reserved_bits_at_4 : 27;
  orderly : 1;
} TPMA_STARTUP_CLEAR;
```
#define TPMA_STARTUP_CLEAR_INITIALIZER(
    phenable, shenable, ehenable, phenablenv, bits_at_4, orderly)

#define TPMA_STARTUP_CLEAR phEnable     ((TPMA_STARTUP_CLEAR)1 << 0)
#define TPMA_STARTUP_CLEAR shEnable     ((TPMA_STARTUP_CLEAR)1 << 1)
#define TPMA_STARTUP_CLEAR ehEnable     ((TPMA_STARTUP_CLEAR)1 << 2)
#define TPMA_STARTUP_CLEAR phEnableNV   ((TPMA_STARTUP_CLEAR)1 << 3)
#define TPMA_STARTUP_CLEAR orderly      ((TPMA_STARTUP_CLEAR)1 << 31)

This implements Table 2:35 TPMA_STARTUP_CLEAR using bit masking.

typedef UINT32                          TPMA_STARTUP_CLEAR;

#define TPMA_MEMORY_INITIALIZER(
    sharedram, sharednv, objectcopiedtoram, bits_at_3)

#define TPMA_MEMORY_sharedRAM           ((TPMA_MEMORY)1 << 0)
#define TPMA_MEMORY_sharedNV            ((TPMA_MEMORY)1 << 1)
#define TPMA_MEMORY_objectCopiedToRam   ((TPMA_MEMORY)1 << 2)
#define TPMA_MEMORY_reserved            ((TPMA_MEMORY)1 << 31)

This implements Table 2:36 TPMA_MEMORY using bit masking.

typedef struct TPMA_MEMORY {
    unsigned sharedRAM            : 1;
    unsigned sharedNV             : 1;
    unsigned objectCopiedToRam    : 1;
    unsigned Reserved_bits_at_3   : 29;
} TPMA_MEMORY;

#define TPMA_CC_initializer(
    commandIndex, Reserved_bits_at_16, nv, extensive, flushed,
    cHandles, rHandle, v, Reserved_bits_at_30)

This implements Table 2:37 TPMA_CC using bit masking.

typedef struct TPMA_CC {
    unsigned commandIndex         : 16;
    unsigned Reserved_bits_at_16  : 6;
    unsigned nv                   : 1;
    unsigned extensive            : 1;
    unsigned flushed              : 1;
    unsigned cHandles             : 3;
    unsigned rHandle              : 1;
    unsigned v                    : 1;
    unsigned Reserved_bits_at_30  : 2;
} TPMA_CC;
#define TPMA_CC_INITIALIZER(commandindex, bits_at_16, nv, extensive, flushed, chandles, rhandle, v, bits_at_30) 

This implements Table 2:37 TPMA_CC using bit masking

typedef UINT32                      TPMA_CC;
#define TPMA_CC_commandIndex_SHIFT  0
#define TPMA_CC_commandIndex        ((TPMA_CC)0xffff << 0)
#define TPMA_CC_nv                  ((TPMA_CC)1 << 22)
#define TPMA_CC_extensive           ((TPMA_CC)1 << 23)
#define TPMA_CC_flushed             ((TPMA_CC)1 << 24)
#define TPMA_CC_cHandles_SHIFT      25
#define TPMA_CC_cHandles            ((TPMA_CC)0x7 << 25)
#define TPMA_CC_rHandle             ((TPMA_CC)1 << 28)
#define TPMA_CC_V                   ((TPMA_CC)1 << 29)

This is the initializer for a TPMA_CC bit array.

#define TPMA_CC_INITIALIZER(commandindex, bits_at_16, nv, extensive, flushed, chandles, rhandle, v, bits_at_30) ((commandindex << 0) + (nv << 22) + (extensive << 23) + (flushed << 24) + (chandles << 25) + (rhandle << 28) + (v << 29))

#endif // USE_BIT_FIELD_STRUCTURES
#if USE_BIT_FIELD_STRUCTURES
typedef struct TPMA_MODES {
    unsigned FIPS_140_2           : 1;
    unsigned Reserved_bits_at_1   : 31;
} TPMA_MODES;
/* Bits */

This is the initializer for a TPMA_MODES structure

#define TPMA_MODES_INITIALIZER(fips_140_2, bits_at_1) {fips_140_2, bits_at_1}

This implements Table 2:38 TPMA_MODES using bit masking

typedef UINT32                  TPMA_MODES;
#define TPMA_MODES_FIPS_140_2   ((TPMA_MODES)1 << 0)

This is the initializer for a TPMA_MODES bit array.

#define TPMA_MODES_INITIALIZER(fips_140_2, bits_at_1) ((fips_140_2 << 0))

#endif // USE_BIT_FIELD_STRUCTURES

typedef BYTE                TPMI_YES_NO;

typedef TPM_HANDLE          TPMI_DH_OBJECT;
typedef TPM_HANDLE          TPMI_DH_PERSISTENT;
typedef TPM_HANDLE          TPMI_DH_ENTITY;
typedef TPM_HANDLE          TPMI_DH_PCR;
typedef TPM_HANDLE          TPMI_SH_AUTH_SESSION;
typedef TPM_HANDLE          TPMI_SH_HMAC;
typedef TPM_HANDLE          TPMI_SH_POLICY;
typedef TPM_HANDLE          TPMI_SH_CONTEXT;
typedef TPM_HANDLE          TPMI_SH_SAVED;
typedef TPM_HANDLE          TPMI_SH_HIERARCHY;
typedef TPM_HANDLE          TPMI_SH_ENABLES;
typedef TPM_HANDLE          TPMI_SH_HIERARCHY_AUTH;

This implements Table 2:39 TPMA_MODES and Table 2:40 TPMA_MODES using bit masking

typedef TPMI_DH_PARENT;
typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:41 TPMA_MODES and Table 2:42 TPMA_MODES using bit masking

typedef TPMI_DH_CONTEXT;
typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:43 TPMA_MODES and Table 2:44 TPMA_MODES using bit masking

typedef TPMI_DH_SAVED;
typedef TPMI_DH_ENABLES;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:45 TPMA_MODES and Table 2:46 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:47 TPMA_MODES and Table 2:48 TPMA_MODES using bit masking

typedef TPMI_DH_CONTEXT;
typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:49 TPMA_MODES and Table 2:50 TPMA_MODES using bit masking

typedef TPMI_DH_PERSISTENT;
typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:51 TPMA_MODES and Table 2:52 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:53 TPMA_MODES and Table 2:54 TPMA_MODES using bit masking

typedef TPMI_DH_PERSISTENT;
typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:55 TPMA_MODES and Table 2:56 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:57 TPMA_MODES and Table 2:58 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:59 TPMA_MODES and Table 2:60 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:61 TPMA_MODES and Table 2:62 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:63 TPMA_MODES and Table 2:64 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:65 TPMA_MODES and Table 2:66 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:67 TPMA_MODES and Table 2:68 TPMA_MODES using bit masking

typedef TPMI_DH_HIERARCHY;
typedef TPMI_DH_HIERARCHY_AUTH;

This implements Table 2:69 TPMA_MODES and Table 2:70 TPMA_MODES using bit masking
typedef TPM_HANDLE          TPMI_RH_PLATFORM; // Table 2:53 /* Interface */
typedef TPM_HANDLE          TPMI_RH_OWNER;  // Table 2:54 /* Interface */
typedef TPM_HANDLE          TPMI_RH_ENDORSEMENT;  // Table 2:55 /* Interface */
typedef TPM_HANDLE          TPMI_RH_PROVISION;  // Table 2:56 /* Interface */
typedef TPM_HANDLE          TPMI_RH_CLEAR;  // Table 2:57 /* Interface */
typedef TPM_HANDLE          TPMI_RH_NV_AUTH;  // Table 2:58 /* Interface */
typedef TPM_HANDLE          TPMI_RH_LOCKOUT;  // Table 2:59 /* Interface */
typedef TPM_HANDLE          TPMI_RH_NV_INDEX;  // Table 2:60 /* Interface */
typedef TPM_HANDLE          TPMI_RH_AC;  // Table 2:61 /* Interface */
typedef TPM_HANDLE          TPMI_RH_PROVISION;  // Table 2:62 /* Interface */
typedef TPM_HANDLE          TPMI_RH_CLEAR;  // Table 2:63 /* Interface */
typedef TPM_HANDLE          TPMI_RH_NV_AUTH;  // Table 2:64 /* Interface */
typedef TPM_HANDLE          TPMI_RH_LOCKOUT;  // Table 2:65 /* Interface */
typedef TPM_HANDLE          TPMI_RH_NV_INDEX;  // Table 2:66 /* Interface */
typedef TPM_HANDLE          TPMI_RH_AC;  // Table 2:67 /* Interface */
typedef TPM_HANDLE          TPMI_RH_PROVISION;  // Table 2:68 /* Interface */
typedef TPM_HANDLE          TPMI_RH_CLEAR;  // Table 2:69 /* Interface */
typedef TPM_HANDLE          TPMI_RH_NV_AUTH;  // Table 2:70 /* Interface */
typedef TPM_HANDLE          TPMI_RH_LOCKOUT;  // Table 2:71 /* Interface */
typedef TPM_HANDLE          TPMI_RH_NV_INDEX;  // Table 2:72 /* Interface */
typedef TPM_HANDLE          TPMI_RH_AC;  // Table 2:73 /* Interface */

typedef TPM сторы
Table 2:74

typedef struct
Table 2:75

typedef union
Table 2:76

typedef union
Table 2:77

typedef struct
Table 2:78

typedef struct
Table 2:79 - Definition of Types for TPM2B_NONCE

typedef TPM2B_DIGEST        TPM2B_NONCE;
typedef TPM2B_DIGEST TPM2B_AUTH;

typedef TPM2B_DIGEST TPM2B_OPERAND;

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[1024];
    } t;
    TPM2B b;
} TPM2B_EVENT; /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[MAX_DIGEST_BUFFER];
    } t;
    TPM2B b;
} TPM2B_MAX_BUFFER; /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[MAX_NV_BUFFER_SIZE];
    } t;
    TPM2B b;
} TPM2B_MAX_NV_BUFFER; /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[MAX_SYM_BLOCK_SIZE];
    } t;
    TPM2B b;
} TPM2B_IV; /* Structure */

typedef union {
    struct {
        TPM_ST tag;
        TPMI_RH_HIERARCHY hierarchy;
    } t;
    TPM2B b;
} TPM2B_NAME; /* Structure */

typedef struct {
    TPM2B_DIGEST digest;
    TPM_HANDLE handle;
} TPMU_NAME; /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE name[sizeof(TPMU_NAME)];
    } t;
    TPM2B b;
} TPM2B_NAME; /* Structure */

typedef struct {
    TPM2B_DIGEST pcrSelect[PCR_SELECT_MAX];
} TPMS_PCR_SELECT; /* Structure */

typedef struct {
    TPMU_NAME handle;
} TPM2B_NAME; /* Structure */

typedef struct {
    TPM2B_DIGEST pcrSelect[PCR_SELECT_MAX];
} TPMS_PCR_SELECTION; /* Structure */
typedef struct {
    TPM2B_DIGEST digest;
} TPMT_TK_CREATION;

typedef struct {
    TPM_ST tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST digest;
} TPMT_TK_VERIFIED;

typedef struct {
    TPM_ST tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST digest;
} TPMT_TK_AUTH;

typedef struct {
    TPM_ST tag;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_DIGEST digest;
} TPMT_TK_HASHCHECK;

typedef struct {
    TPM_ST                  tag;
    TPMI_RH_HIERARCHY       hierarchy;
    TPM2B_DIGEST            digest;
} TPMT_TKHASHCHECK;

typedef struct {
    TPM_ST                  tag;
    TPMI_RH_HIERARCHY       hierarchy;
    TPM2B_DIGEST            digest;
} TPMT_TK_AUTH;

typedef struct {
    TPM_ST                  tag;
    TPMI_RH_HIERARCHY       hierarchy;
    TPM2B_DIGEST            digest;
} TPMT_TK_HASHCHECK;

typedef struct {
    TPM_ALG_ID              alg;
    TPMA_ALGORITHM          algProperties;
} TPMS_ALG_PROPERTY;

typedef struct {
    TPM_PT                  property;
    UINT32                  value;
} TPMS_TAGGED_PROPERTY;

typedef struct {
    TPM_PT_PCR              tag;
    UINT8                   sizeofSelect;
    BYTE                    pcrSelect[PCR_SELECT_MAX];
} TPMS_TAGGED_PCR_SELECT;

typedef struct {
    TPM_HANDLE              handle;
    TPMT_HA                 policyHash;
} TPMS_TAGGED_POLICY;

typedef struct {
    UINT32                  count;
    TPM_CC                  commandCodes[MAX_CAP_CC];
} TPML_CC;

typedef struct {
    UINT32                  count;
    TPM_ALG_ID              algorithms[MAX_ALG_LIST_SIZE];
} TPML_ALG;

typedef struct {
    UINT32                  count;
    TPM_HANDLE              handle[MAX_CAP_HANDLES];
} TPML_HANDLE;

typedef struct {
    UINT32                  count;
    TPM2B_DIGEST            digests[8];
} TPML_DIGEST;

typedef struct {
    UINT32                  count;
    TPMT_HA                 digests[HASH_COUNT];
} TPML_DIGEST_VALUES;

typedef struct {
    UINT32                  count;
    TPM_ALG_ID              algProperties[MAX_CAP_ALGS];
} TPMS_ALG_PROPERTY;

typedef struct {
    TPM_ID                  id;
    TPM_ALG_LIST           algList;
} TPMS_ALG_LIST;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_PROVISION;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_VOUCHER;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_CONFIRM;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_CREDENTIAL;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_PUBLICATION;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_PUBLIC;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_PRIVATE;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_SIGNED;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_VERIFIED;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_AUTH;

typedef struct {
    TPM_ST                  tag;
    TPM2B_DIGEST            digest;
} TPMT_TK_HASHCHECK;
typedef struct {                      // Table 2:109
    UINT32 count;
    TPM_TAGGED_PROPERTY tpmProperty[MAX_TPM_PROPERTIES];
} TPML_TAGGED_TPM_PROPERTY; /* Structure */

typedef struct {                  // Table 2:110
    UINT32 count;
    TPM_TAGGED_PCR_SELECT pcrProperty[MAX_PCR_PROPERTIES];
} TPML_TAGGED_PCR_PROPERTY; /* Structure */

typedef struct {              // Table 2:111
    UINT32 count;
    TPM_ECC_CURVE eccCurves[MAX_ECC_CURVES];
} TPML_ECC_CURVE; /* Structure */

typedef struct {          // Table 2:112
    UINT32 count;
    TPM_TAGGED_POLICY policies[MAX_TAGGED_POLICIES];
} TPML_TAGGED_POLICY; /* Structure */

typedef union {        // Table 2:113
    TPML_ALG_PROPERTY algorithms;
    TPML_HANDLE handles;
    TPML_CCA command;
    TPML_CC ppCommands;
    TPML_CC auditCommands;
    TPML_PCR_SELECTION assignedPCR;
    TPML_TAGGED_TPM_PROPERTY tpmProperties;
    TPML_TAGGED_PCR_PROPERTY pcrProperties;
    #if ALG_ECC
    TPML_ECC_CURVE eccCurves;
    #endif // ALG_ECC
    TPML_TAGGED_POLICY authPolicies;
} TPMU_CAPABILITIES; /* Structure */

typedef struct { // Table 2:114
    TPM_CAP capability;
    TPMU_CAPABILITIES data;
} TPMS_CAPABILITY_DATA; /* Structure */

typedef struct { // Table 2:115
    UINT64 clock;
    TPMU_CAPABILITIES data;
} TPMS_CLOCK_INFO; /* Structure */

typedef struct { // Table 2:116
    TPM2B_DIGEST pcrDigest;
    TPM2B_DIGEST commandDigest;
} TPMS_QUOTE_INFO; /* Structure */

typedef struct { // Table 2:117
    TPM2B_DIGEST sessionDigest;
    TPM2B_DIGEST commandDigest;
} TPMS_SESSION_AUDIT_INFO; /* Structure */
typedef struct {
    TPM2B_NAME objectName;
    TPM2B_DIGEST creationHash;
} TPMS_CREATION_INFO; /* Structure */

typedef struct {
    TPM2B_NAME indexName;
    UINT16 offset;
    TPM2B_MAX_NV_BUFFER nvContents;
} TPMS_NV_CERTIFY_INFO; /* Structure */

typedef TPMS_CREATION_INFO TPMS_CREATION_INFO; /* Interface */

typedef struct {
    TPM_GENERATED magic;
    TPMI_ST_ATTEST type;
    TPM2B_NAME qualifiedSigner;
    TPM2B_DATA extraData;
    UINT64 clockInfo;
    TPMU_ATTEST attested;
} TPMS_ATTEST; /* Structure */

typedef struct {
    UINT16 size;
    BYTE attestationData[sizeof(TPMS_ATTEST)];
} t;

typedef struct {
    TPM2B_NONCE nonce;
    TPM2B_SESSION sessionAttributes;
    TPM2B_AUTH hmac;
} TMPS_AUTH_COMMAND; /* Structure */

typedef struct {
    TPM2B_NONCE nonce;
    TPM2B_SESSION sessionAttributes;
    TPM2B_AUTH hmac;
} TMPS_AUTH_RESPONSE; /* Structure */

typedef TPM_KEY_BITS TPMS_TDES_KEY_BITS; /* Interface */

typedef TPM_KEY_BITS TPMS_AES_KEY_BITS; /* Interface */

typedef TPM_KEY_BITS TPMS_SM4_KEY_BITS; /* Interface */

if ALG_TDES
    TPMS_TDES_KEY_BITS tdes;
endif; // ALG_TDES

if ALG_AES
    TPMS_AES_KEY_BITS aes;
endif; // ALG_AES

if ALG_SM4
    TPMS_SM4_KEY_BITS sm4;
endif; // ALG_SM4

if ALG_CAMELLIA
    TPMS_CAMELLIA_KEY_BITS camellia;
endif; // ALG_CAMELLIA

if ALG_XOR
    TPM_KEY_BITS sym;
endif; // ALG_XOR

TPMS_CAMELLIA_KEY_BITS xor;
1019 #endif
1020 } TPMU_SYM_KEY_BITS;  /* Structure */
1021 typedef union {
1022   #if
1023     TPMI_ALG_SYM_MODE tdes;
1024   #endif
1025   #if
1026     TPMI_ALG_SYM_MODE aes;
1027   #endif
1028   #if
1029     TPMI_ALG_SYM_MODE sm4;
1030   #endif
1031   #if
1032     TPMI_ALG_SYM_MODE camellia;
1033   #endif
1034   TPMI_ALG_SYM_MODE sym;
1035 } TPMU_SYM_MODE;  /* Structure */
1036 typedef struct {
1037   TPMI_ALG_SYM algorithm;
1038   TPMU_SYM_KEY_BITS keyBits;
1039   TPMI_ALG_SYM_MODE mode;
1040 } TPMU_SYM_DEF;  /* Structure */
1041 typedef struct {
1042   TPMI_ALG_SYM_OBJECT algorithm;
1043   TPMU_SYM_KEY_BITS keyBits;
1044   TPMI_ALG_SYM_MODE mode;
1045 } TPMU_SYM_DEF_OBJECT;  /* Structure */
1046 typedef union {
1047   struct {
1048     UINT16 size;
1049     BYTE buffer[MAX_SYM_KEY_BYTES];
1050   } t;
1051   TPM2B b;
1052 } TPM2B_SYM_KEY;  /* Structure */
1053 typedef struct {
1054   TPMU_SYM_DEF Object;
1055 } TPMU_SYM_DEF_OBJECT;  /* Structure */
1056 typedef union {
1057   struct {
1058     TPM2B_SYM_KEY;  /* Structure */
1059   } t;
1060   TPM2B b;
1061 } TPM2B_LABEL;  /* Structure */
1062 typedef struct {
1063   TPMU_SYM_DEF OBJECT;
1064   TPMU_SYM_DEF_OBJECT;
1065 } TPMU_SENSITIVE_CREATE;  /* Structure */
1066 typedef union {
1067   struct {
1068     TPM2B_LABEL;  /* Structure */
1069   } t;
1070   TPM2B b;
1071 } TPM2B_DERIVE;  /* Structure */
1072 typedef union {
1073   struct {
1074     TPMU_SENSITIVE_CREATE;  /* Structure */
1075   } t;
1076   TPM2B b;
1077 } TPMU_SENSITIVE_CREATE;  /* Structure */
1078 typedef union {
1079   struct {
1080     TPMU_SENSITIVE_DATA;  /* Structure */
1081   } t;
1082   TPM2B b;
1083 } TPMU_SENSITIVE_DATA;  /* Structure */
typedef struct {   // Table 2:143
    TPM2B_AUTH userAuth;
    TPM2B_SENSITIVE_DATA data;
} TPMS_SENSITIVE_CREATE;  /* Structure */

typedef struct {   // Table 2:144
    UINT16 size;
} TPMS_SENSITIVE_CREATE;  /* Structure */

typedef struct {   // Table 2:145
    TPMI_ALG_HASH hashAlg;
} TPMS_SCHEME_HASH;  /* Structure */

typedef struct {   // Table 2:146
    TPMI_ALG_HASH hashAlg;
    UINT16 count;
} TPMS_SCHEME_ECDAA;  /* Structure */

typedef TPM_ALG_ID TPMI_ALG_KEYEDHASH_SCHEME;

Table 2:148 - Definition of Types for HMAC_SIG_SCHEME

typedef TPMS_SCHEME_HASH TPMS_SCHEME_HMAC;  // Table 2:149

typedef struct {   // Table 2:149
    TPMI_ALG_HASH hashAlg;
    TPMI_ALG_KDF kdf;
} TPMS_SCHEME_XOR;  /* Structure */

typedef union {   // Table 2:150
    #if ALG_HMAC
    TPMS_SCHEME_HMAC hmac;
    #endif  // ALG_HMAC
    #if ALG_XOR
    TPMS_SCHEME_XOR xor;
    #endif  // ALG_XOR
} TPMU_SCHEME_KEYEDHASH;  /* Structure */

typedef struct {   // Table 2:151
    TPMI_ALG_KEYEDHASH_SCHEME scheme;
    TPMU_SCHEME_KEYEDHASH details;
} TPMT_KEYEDHASH_SCHEME;  /* Structure */

Table 2:152 - Definition of Types for RSA Signature Schemes

typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_RSASSA;

typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_RSAPSS;

Table 2:153 - Definition of Types for ECC Signature Schemes

typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_ECDSA;

typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_SM2;

typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_ECSCHNORR;

typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_ECSCHNORR;

typedef TPMS_SCHEME_HASH TPMS_SIG_SCHEME_ECSCHNORR;

typedef union {   // Table 2:154
    #if ALG_ECC
    TPMS_SIG_SCHEME_ECDAA ecdaa;
    #endif  // ALG_ECC
    #if ALG_RSASSA
    TPMS_SIG_SCHEME_RSASSA rsassa;
    #endif  // ALG_RSASSA
    #if ALG_RSAPSS
    TPMS_SIG_SCHEME_RSAPSS rsapss;
    #endif  // ALG_RSAPSS
    #if ALG_ECDSA
    TPMS_SIG_SCHEME_ECDAA ecdsa;
    #endif  // ALG_ECDSA
    #if ALG_SM2
    TPMS_SIG_SCHEME_SM2 sm2;
    #endif  // ALG_SM2

```c
#define ALG_ECSCHNORR
TPM_SIG_SCHEME_ECSCHNORR ecschnorr;
#endif // ALG_ECSCHNORR
#define ALG_HMAC
TPM_SCHEME_HMAC hmac;
#endif // ALG_HMAC
#define ALG_RSAPSS
TPM_SIG_SCHEME_RSAPSS rsapss;
#endif // ALG_RSAPSS
typedef struct {     // Table 2:155
    TPMI_ALG_SIG_SCHEME scheme;
    TPMU_SIG_SCHEME details;
} TPMT_SIG_SCHEME;   /* Structure */

Table 2:156 - Definition of Types for Encryption Schemes

typedef TPM_SCHEMA_HASH    TPMS_ENC_SCHEME_OAEP;
typedef TPM_SCHEMA_EMPTY   TPMS_ENC_SCHEME_RSAES;

Table 2:157 - Definition of Types for ECC Key Exchange

typedef TPM_SCHEMA_HASH    TPMS_KEY_SCHEME_ECDH;
typedef TPM_SCHEMA_HASH    TPMS_KEY_SCHEME_ECMQV;

typedef TPMS_SCHEMA_HASH         TPMS_SCHEMA_MGF1;
typedef TPM_SCHEMA_HASH         TPMS_SCHEMA_KDF1_SP800_56A;
typedef TPM_SCHEMA_HASH         TPMS_SCHEMA_KDF2;
typedef TPM_SCHEMA_HASH         TPMS_SCHEMA_KDF1_SP800_108;
tyepdef union {     // Table 2:159
    TPM_SCHEMA_MGF1 mgf1;
#endif // ALG_MGF1
    TPM_SCHEMA_KDF1_sp800_56A kdf1_sp800_56a;
#endif // ALG_KDF1_SP800_56A
    TPM_SCHEMA_KDF2 kdf2;
#endif // ALG_KDF2
    TPM_SCHEMA_KDF1_sp800_108 kdf1_sp800_108;
#endif // ALG_KDF1_SP800_108
} TPMU_KDF_SCHEME;   /* Structure */

typedef struct {     // Table 2:160
    TPMI_ALG_KDF scheme;
    TPMU_KDF_SCHEME details;
} TPMT_KDF_SCHEME;   /* Structure */

typedef TPM_ALG_ID          TPMI_ALG_ASYM_SCHEME;
// Table 2:161  /* Interface */

#define ALG_ECDH
TPM_KEY_SCHEME_ECDH ecdh;
#endif // ALG_ECDH
#define ALG_ECC
TPM_KEY_SCHEME_ECC ecmq;
#endif // ALG_ECMQV
#endif // ALG_ECC
#define ALG_RSASSA
TPM_SIG_SCHEME_RSASSA rsassa;
#endif // ALG_RSAPSS
```

Table 2:158 - Definition of Types for KDF Schemes

```
```
```c
#define     #if ALG_ECDSA
TPMS_SIG_SCHEME_ECDSA ecdsa;
#endif // ALG_ECDSA
#define     #if ALG_SM2
TPMS_SIG_SCHEME_SM2 sm2;
#endif // ALG_SM2
#define     #if ALG_ECSCHNORR
TPMS_SIG_SCHEME_ECSCHNORR ecschnorr;
#endif // ALG_ECSCHNORR
#define     #if ALG_RSAES
TPMS_ENC_SCHEME_RSAES rsaes;
#endif // ALG_RSAES
#define     #if ALG_OAEP
TPMS_ENC_SCHEME_OAEP oaep;
#endif // ALG_OAEP
#endif // TPMU_ASYM_SCHEME;
/* Structure */
typedef struct {
    TPMI_ALG_ASYM_SCHEME scheme;
    TPMU_ASYM_SCHEME details;
} TPMT_ASYM_SCHEME;
/* Structure */
typedef TPM_ALG_ID TPMI_ALG_RSA_SCHEME;
// Table 2:164  /* Interface */
typedef struct {
    TPMI_ALG_RSA_SCHEME scheme;
    TPMU_ASYM_SCHEME details;
} TPMT_RSA_SCHEME;
/* Structure */
typedef TPM_ALG_ID TPMI_ALG_RSA_DECRYPT;
// Table 2:166  /* Interface */
typedef struct {
    TPMI_ALG_RSA_DECRYPT scheme;
    TPMU_ASYM_SCHEME details;
} TPMT_RSA_DECRYPT;
/* Structure */
typedef union {
    struct {
        UINT16 size;
        BYTE   buffer[MAX_RSA_KEY_BYTES];
    } t;
    TPM2B     b;
} TPM2B_PUBLIC_KEY_RSA;
/* Structure */
typedef TPM_KEY_BITS TPMI_KEY_BITS; // Table 2:169 /* Interface */
typedef union {
    struct {
        UINT16 size;
        BYTE   buffer[MAX_RSA_KEY_BYTES/2];
    } t;
    TPM2B     b;
} TPM2B_PRIVATE_KEY_RSA;
/* Structure */
typedef union {
    struct {
        UINT16 size;
        BYTE   buffer[MAX_ECC_KEY_BYTES];
    } t;
    TPM2B     b;
} TPM2B_ECC_PARAMETER;
/* Structure */
typedef struct {
    TPM2B_ECC_PARAMETER x;
    TPM2B_ECC_PARAMETER y;
} TPMS_ECC_POINT;
/* Structure */
typedef struct {
    UINT16 size;
    TPMS_ECC_POINT point;
} TPMS_ECC_POINT;
/* Structure */
typedef struct {
    TPMI_ALG_ECC_SCHEME scheme;
} TPMU_ASYM_SCHEME details;
```
1261 ) TPMT_ECC_SCHEME; /* Structure */
1262 typedef struct {
1263   TPM_ECC_CURVE curveID;
1264   UINT16      keySize;
1265   TPMT_KDF_SCHEME kdf;
1266   TPMT_ECC_SCHEME sign;
1267   TPMT2B_ECC_PARAMETER p;
1268   TPMT2B_ECC_PARAMETER a;
1269   TPMT2B_ECC_PARAMETER b;
1270   TPMT2B_ECC_PARAMETER gX;
1271   TPMT2B_ECC_PARAMETER gY;
1272   TPMT2B_ECC_PARAMETER n;
1273   TPMT2B_ECC_PARAMETER h;
1274 } TPMS_ALGORITHM_DETAIL_ECC; /* Structure */
1275 typedef struct {
1276   TPMI_ALG_HASH hash;
1277   TPM2B_PUBLIC_KEY_RSA sig;
1278 } TPMS_SIGNATURE_RSA; /* Structure */

Table 2:179 - Definition of Types for Signature
1279 typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSASSA;
1280 typedef TPMS_SIGNATURE_RSA TPMS_SIGNATURE_RSAPSS;
1281 typedef struct {
1282   TPMI_ALG_HASH hash;
1283   TPM2B_ECC_PARAMETER signatureR;
1284   TPM2B_ECC_PARAMETER signatureS;
1285 } TPMS_SIGNATURE_ECC; /* Structure */

Table 2:181 - Definition of Types for TPMS_SIGNATURE_ECC
1286 typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDAA;
1287 typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECDSA;
1288 typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_SM2;
1289 typedef TPMS_SIGNATURE_ECC TPMS_SIGNATURE_ECSCHNORR;
1290 typedef union {
1291   #if ALG_ECC
1292     TPMS_SIGNATURE_ECDAA ecdaa;
1293     #endif // ALG_ECC
1294   #if ALG_RSA
1295     TPMS_SIGNATURE_RSASSA rsassa;
1296     #endif // ALG_RSA
1297   #if ALG_RSA
1298     TPMS_SIGNATURE_RSAPSS rsapss;
1299     #endif // ALG_RSA
1300   #if ALG_ECC
1301     TPMS_SIGNATURE_ECDSA ecdsa;
1302     #endif // ALG_ECC
1303   #if ALG_ECC
1304     TPMS_SIGNATURE_SM2 sm2;
1305     #endif // ALG_ECC
1306   #if ALG_ECC
1307     TPMS_SIGNATURE_ECSCHNORR ecschnorr;
1308     #endif // ALG_ECC
1309   #if ALG_HMAC
1310     TPM_HA hmac;
1311     #endif // ALG_HMAC
1312     TPM_SCHEME_HASH any;
1313 } TPMU_SIGNATURE; /* Structure */
1314 typedef struct {
1315   TPMI_ALG_SIG_SCHEME sigAlg;
1316   TPMU_SIGNATURE signature;
1317 } TPMT_SIGNATURE; /* Structure */
1318 typedef union {
1319   #if ALG_ECC
1320 #endif
BYTE ecc[sizeof(TPMS_ECC_POINT)];

if (ALG_ECC)
BYTE rsa[MAX_RSA_KEY_BYTES];
if (ALG_RSA)
BYTE symmetric[sizeof(TPM2B_DIGEST)];
if (ALG_SYMCIPHER)
BYTE keyedHash[sizeof(TPM2B_DIGEST)];
if (ALG_KEYEDHASH)

typedef union {
  struct {
    UINT16 size;
    BYTE secret[sizeof(TPMU_ENCRYPTED_SECRET)];
  } t;
  TPM2B b;
} TPMU_ENCRYPTED_SECRET;
/* Structure */

typedef union {
  struct {
    TPM2B_DIGEST keyedHash;
    TPM2B_PUBLIC_KEY_RSA rsa;
    TPM2B_PUBLIC_KEY_ECC ecc;
  } TPMU_PUBLIC_ID;
/* Structure */

typedef struct {
  TPM_ALG_ID TPMI_ALG_PUBLIC;
  TPMU_PUBLIC_ID;
  TPMU_PUBLIC_ID;
/* Interface */

typedef union {
  struct {
    TPMT_KEYEDHASH_SCHEME scheme;
    TPMT_SYMCIPHER_PARMS symmetric;
    TPMT_RSA_PARMS rsa;
  } TPMU_PUBLIC_ID;
/* Structure */

typedef struct {
  TPMT_SYM_DEF_OBJECT symmetric;
  TPMT_SYM_DEF_OBJECT symmetric;
  TPMT_SYMCIPHER_SCHEME scheme;
  TPMT_RSA_SCHEME scheme;
  TPMT_ECC_SCHEME scheme;
  TPM2B_DIGEST scheme;
  TPM2B_DIGEST scheme;
  TPM2B_DIGEST scheme;
/* Structure */

typedef struct {
  TPMT_SYM_DEF_OBJECT symmetric;
  TPMT_RSA_SYM_SIG_SCHEME scheme;
  TPM2B_DIGEST scheme;
  TPM2B_DIGEST scheme;
  TPM2B_DIGEST scheme;
/* Structure */

typedef union {
  struct {
    TPMS_ECC_PARMS ecc;
    TPMS_ECC_PARMS ecc;
    TPMS_ECC_PARMS ecc;
/* Structure */

typedef union {
  struct {
    TPMS_KEYEDHASH_PARMS keyedHashDetail;
    TPMS_KEYEDHASH_PARMS keyedHashDetail;
    TPMS_KEYEDHASH_PARMS keyedHashDetail;
/* Structure */

Page 172  TCG Public Review  Family “2.0”
September 18, 2018  Copyright © TCG 2006-2019  Level 00 Revision 01.50

DRAFT
/* Structure */
typedef struct {
    TPMI_ALG_PUBLIC type;
    TPMI_ALG_HASH nameAlg;
    TPM_A_OBJECT objectAttributes;
    TPM2B_DIGEST authPolicy;
    TPMU_PUBLIC_PARMS parameters;
    TPMU_PUBLIC_ID unique;
} TPMT_PUBLIC; /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[sizeof(TPMT_PUBLIC)];
    } t;
    TPM2B b;
} TPM2B_TEMPLATE; /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[PRIVATE_VENDOR_SPECIFIC_BYTES];
    } t;
    TPM2B b;
} TPM2B_PRIVATE_VENDOR_SPECIFIC; /* Structure */

typedef union {
    #if ALG_RSA
    TPM2B_PUBLIC RSA                rsa;
    #endif // ALG_RSA
    #if ALG_ECC
    TPM2B_PUBLIC ECC_PARAMETER       ecc;
    #endif // ALG_ECC
    #if ALG_KEYEDHASH
    TPM2B_PUBLIC SENSITIVE DATA     bits;
    #endif // ALG_KEYEDHASH
    #if ALG_SYMCIPHER
    TPM2B_PUBLIC SYM_KEY            sym;
    #endif // ALG_SYMCIPHER
    #if ALG_KEYEDHASH
    TPM2B_PUBLIC SENSITIVE Data     authValue;
    #endif // ALG_KEYEDHASH
    #if ALG_SYMCIPHER
    TPM2B_PUBLIC SYM_KEY            seedValue;
    #endif // ALG_SYMCIPHER
    #if ALG_ECC
    TPM2B_PUBLIC ECC_PARAMETER      parameters;
    #endif // ALG_ECC
} TPMU_PUBLIC_PARMS; /* Structure */

typedef struct {
    TPMI_ALG_PUBLIC type;
    TPMI_ALG_HASH nameAlg;
    TPM_A_OBJECT objectAttributes;
    TPM2B_DIGEST authPolicy;
    TPMU_PUBLIC_PARMS parameters;
    TPMU_PUBLIC_ID unique;
} TPMU_PUBLIC; /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[PRIVATE_VENDOR_SPECIFIC_BYTES];
    } t;
    TPM2B b;
} TPM2B_PRIVATE_VENDOR_SPECIFIC; /* Structure */

typedef struct {
    TPMI_ALG_PUBLIC sensitiveType;
    TPM2B_AUTH authValue;
    TPM2B_DIGEST seedValue;
    TPMU_PUBLIC_PARMS parameters;
} TPMT_SENSITIVE; /* Structure */

typedef struct {
    UINT16 size;
    TPMU_PUBLIC PARMS sensitive;
} TPMT_PUBLIC; /* Structure */

typedef struct {
    TPMI_ALG_PUBLIC type;
    TPMI_ALG_HASH nameAlg;
    TPM_A_OBJECT objectAttributes;
    TPM2B_DIGEST authPolicy;
    TPMU_PUBLIC_PARMS parameters;
    TPMU_PUBLIC_ID unique;
} TPMU_PUBLIC; /* Structure */

typedef struct {
    TPMI_ALG_PUBLIC type;
    TPMI_ALG_HASH nameAlg;
    TPM_A_OBJECT objectAttributes;
    TPM2B_DIGEST authPolicy;
    TPMU_PUBLIC_PARMS parameters;
    TPMU_PUBLIC_ID unique;
} TPMU_PUBLIC; /* Structure */

typedef struct {
    TPMI_ALG_PUBLIC type;
    TPMI_ALG_HASH nameAlg;
    TPM_A_OBJECT objectAttributes;
    TPM2B_DIGEST authPolicy;
    TPMU_PUBLIC_PARMS parameters;
    TPMU_PUBLIC_ID unique;
} TPMU_PUBLIC; /* Structure */
1452     BYTE                buffer[sizeof(_PRIVATE)];
1453     } t;
1454     TPM2B b;
1455 } TPM2B_PRIVATE;           /* Structure */
1456 typedef struct {
1457     TPM2B_DIGEST integrityHMAC;
1458     TPM2B_DIGEST encIdentity;
1459 } TPM2B_PRIVATE;           /* Structure */
1460 typedef union {
1461     struct {
1462     UINT16 size;
1463     BYTE credential[sizeof(TPMS_ID_OBJECT)];
1464     } t;
1465     TPM2B b;
1466 } TPM2B_ID_OBJECT;         /* Structure */
1467 #if USE_BIT_FIELD_STRUCTURES
1468 typedef struct TPM_NV_INDEX {
1469     unsigned index : 24;
1470     unsigned RH_NV : 8;
1471 } TPM_NV_INDEX;            /* Bits */
1472 #define TPM_NV_INDEX_INITIALIZER(index, rh_nv) {index, rh_nv}
1473 #else // USE_BIT_FIELD_STRUCTURES
1474 typedef UINT32 TPM_NV_INDEX;
1475 #define TPM_NV_INDEX_INDEX_SHIFT 0
1476 #define TPM_NV_INDEX_INDEX ((TPM_NV_INDEX)0xffffffff << 0)
1477 #define TPM_NV_INDEX_RH_NV_SHIFT 24
1478 #define TPM_NV_INDEX_RH_NV ((TPM_NV_INDEX)0xff << 24)
1479 #define TPM_NV_INDEX_INITIALIZER(index, rh_nv) ((index << 0) + (rh_nv << 24))
1480 #endif // USE_BIT_FIELD_STRUCTURES
Table 2:206 - Definition of TPM_NT Constants
1481 typedef UINT32 TPM_NT;
1482 #define TPM_NT_ORDINARY     (TPM_NT)(0x0)
1483 #define TPM_NT_COUNTER      (TPM_NT)(0x1)
1484 #define TPM_NT_BITS         (TPM_NT)(0x2)
1485 #define TPM_NT_EXTEND       (TPM_NT)(0x4)
1486 #define TPM_NT_PIN_FAIL     (TPM_NT)(0x8)
1487 #define TPM_NT_PIN_PASS     (TPM_NT)(0x9)
1488 typedef struct {
1489     UINT32 pinCount;
1490     UINT32 pinLimit;
1491 } TPMS_NV_PIN_COUNTER_PARAMETERS;          /* Structure */
1492 #if USE_BIT_FIELD_STRUCTURES
1493 typedef struct TPM_NV {
1494     unsigned PPWRITE       : 1;
1495     unsigned OWNERWRITE   : 1;
1496     unsigned AUTHWRITE    : 1;
1497     unsigned POLICYWRITE  : 1;
1498     unsigned TPM_NT      : 4;
1499     unsigned Reserved_bits_at_8 : 2;
1500     unsigned POLICY_DELETE       : 1;
1501     unsigned WRITELOCKED : 1;
1502     unsigned WRITEALL    : 1;
1503     unsigned WRITEDEFINE : 1;

This is the initializer for a TPM_NV_INDEX structure

This implements Table 2:205 TPM_NV_INDEX using bit masking

This is the initializer for a TPM_NV_INDEX bit array.
This is the initializer for a TPMA_NV structure:

```c
#define TPMA_NV_INITIALIZER(
    ppwrite, ownerwrite, authwrite, policywrite,
    tpm_nt, bits_at_8, policy_delete, writelocked,
    writeall, writedefine, write_stclear, globallock,
    ppread, ownerread, authread, policyread,
    bits_at_20, no_da, orderly, clear_stclear,
    readlocked, written, platformcreate, read_stclear)
```

This implements Table 2:208 TPMA_NV using bit masking:

```c
typedef UINT32 TPMA_NV;
#define TPMA_NV_PPWRITE ((TPMA_NV)1 << 0)
#define TPMA_NV_OWNERWRITE ((TPMA_NV)1 << 1)
#define TPMA_NV_AUTHWRITE ((TPMA_NV)1 << 2)
#define TPMA_NV_POLICYWRITE ((TPMA_NV)1 << 3)
#define TPMA_NV_TPM_NT_SHIFT 4
#define TPMA_NV_TPM_NT ((TPMA_NV)0xf << 4)
#define TPMA_NV_POLICY_DELETE ((TPMA_NV)1 << 10)
#define TPMA_NV_WRITELOCKED ((TPMA_NV)1 << 11)
#define TPMA_NV_WRITEALL ((TPMA_NV)1 << 12)
#define TPMA_NV_WRITEDEFINE ((TPMA_NV)1 << 13)
#define TPMA_NV_WRITE_STCLEAR ((TPMA_NV)1 << 14)
#define TPMA_NV_GLOBALLOCK ((TPMA_NV)1 << 15)
#define TPMA_NV_PPREAD ((TPMA_NV)1 << 16)
#define TPMA_NV_OWNERREAD ((TPMA_NV)1 << 17)
#define TPMA_NV_AUTHREAD ((TPMA_NV)1 << 18)
#define TPMA_NV_POLICYREAD ((TPMA_NV)1 << 19)
#define TPMA_NV_NO_DA ((TPMA_NV)1 << 25)
#define TPMA_NV_ORDERLY ((TPMA_NV)1 << 26)
#define TPMA_NV_CLEAR_STCLEAR ((TPMA_NV)1 << 27)
#define TPMA_NV_READONLY ((TPMA_NV)1 << 28)
#define TPMA_NV_WRITTEN ((TPMA_NV)1 << 29)
#define TPMA_NV_PLATFORMCREATE ((TPMA_NV)1 << 30)
#define TPMA_NV_READ_STCLEAR ((TPMA_NV)1 << 31)
```

This is the initializer for a TPMA_NV bit array:

```c
#define TPMA_NV_INITIALIZER(
    ppwrite, ownerwrite, authwrite, policywrite,
```
typedef struct { // Table 2:209
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_ALG_HASH nameAlg;
    TPMA_NV attributes;
    TPM2B_DIGEST authPolicy;
    UINT16 dataSize;
} TPMS_NV_PUBLIC;  /* Structure */

typedef struct { // Table 2:210
    UINT16 size;
    TPMS_NV_PUBLIC nvPublic;
} TPM2B_NV_PUBLIC;  /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[MAX_CONTEXT_SIZE];
    } t;
    TPM2B b;
} TPM2B_CONTEXT_SENSITIVE;  /* Structure */

typedef struct { // Table 2:212
    TPM2B_DIGEST integrity;
    TPM2B_CONTEXT_SENSITIVE encrypted;
} TPM2_CTXDATA;  /* Structure */

typedef struct { // Table 2:213
    struct {
        UINT16 size;
        BYTE buffer[sizeof(TPMS_CTXDATA)];
    } t;
    TPM2B b;
} TPM2_CONTEXT;  /* Structure */

typedef struct { // Table 2:214
    UINT64 sequence;
    TPMI_DH_SAVED savedHandle;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_CONTEXT_DATA contextBlob;
} TPMS_CTX;  /* Structure */

typedef struct { // Table 2:216
    TPML_PCR_SELECTION pcrSelect;
    TPM2B_DIGEST pcrDigest;
    TPMA_LOCALITY locality;
    TPM_ALG_ID parentNameAlg;
    TPM2B_NAME parentName;
    TPM2B_NAME parentQualifiedName;
    TPM2B_DATA outsideInfo;
} TPMS_CREATION_DATA;  /* Structure */

typedef struct { // Table 2:217
    UINT16 size;
} TPM2_CREATION_DATA;  /* Structure */

typedef struct { // Table 2:218
    TPM2B_DIGEST pcrDigest;
    TPM_ALG_ID parentNameAlg;
    TPM2B_NAME parentName;
    TPM2B_NAME parentQualifiedName;
    TPM2B_DATA outsideInfo;
} TPMS_CREATION_DATA;  /* Structure */

typedef struct { // Table 2:219
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_ALG_HASH nameAlg;
    TPMA_NV attributes;
    TPM2B_DIGEST authPolicy;
    UINT16 dataSize;
} TPMS_NV_PUBLIC;  /* Structure */

typedef struct { // Table 2:220
    UINT16 size;
    TPMS_NV_PUBLIC nvPublic;
} TPM2B_NV_PUBLIC;  /* Structure */

typedef union {
    struct {
        UINT16 size;
        BYTE buffer[sizeof(TPMS_CTXDATA)];
    } t;
    TPM2B b;
} TPM2B_CONTEXT_DATA;  /* Structure */

typedef struct { // Table 2:221
    UINT64 sequence;
    TPMI_DH_SAVED savedHandle;
    TPMI_RH_HIERARCHY hierarchy;
    TPM2B_CTXDATA contextBlob;
} TPMS_CTX;  /* Structure */

typedef struct { // Table 2:222
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_ALG_HASH nameAlg;
    TPMA_NV attributes;
    TPM2B_DIGEST authPolicy;
    UINT16 dataSize;
} TPMS_NV_PUBLIC;  /* Structure */

typedef struct { // Table 2:223
    UINT16 size;
} TPM2_CTXDATA;  /* Structure */

typedef struct { // Table 2:224
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_ALG_HASH nameAlg;
    TPMA_NV attributes;
    TPM2B_DIGEST authPolicy;
    UINT16 dataSize;
} TPMS_NV_PUBLIC;  /* Structure */

typedef struct { // Table 2:225
    UINT16 size;
} TPM2_CTXDATA;  /* Structure */

typedef struct { // Table 2:226
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_ALG_HASH nameAlg;
    TPMA_NV attributes;
    TPM2B_DIGEST authPolicy;
    UINT16 dataSize;
} TPMS_NV_PUBLIC;  /* Structure */

typedef struct { // Table 2:227
    UINT16 size;
} TPM2_CTXDATA;  /* Structure */

typedef struct { // Table 2:228
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_ALG_HASH nameAlg;
    TPMA_NV attributes;
    TPM2B_DIGEST authPolicy;
    UINT16 dataSize;
} TPMS_NV_PUBLIC;  /* Structure */

typedef struct { // Table 2:229
    UINT16 size;
} TPM2_CTXDATA;  /* Structure */

typedef struct { // Table 2:230
    TPMI_RH_NV_INDEX nvIndex;
    TPMI_ALG_HASH nameAlg;
    TPMA_NV attributes;
    TPM2B_DIGEST authPolicy;
    UINT16 dataSize;
} TPMS_NV_PUBLIC;  /* Structure */

typedef struct { // Table 2:231
    UINT16 size;
} TPM2_CTXDATA;  /* Structure */
Table 2:218 - Definition of TPM_AT Constants

1624  typedef UINT32             TPM_AT;
1625  #define TPM_AT_ANY          (TPM_AT)(0x00000000)
1626  #define TPM_AT_ERROR        (TPM_AT)(0x00000001)
1627  #define TPM_AT_PV1          (TPM_AT)(0x00000002)
1628  #define TPM_AT_VEND         (TPM_AT)(0x80000000)

Table 2:219 - Definition of TPM_AE Constants

1629  typedef UINT32             TPM_AE;
1630  #define TPM_AE_NONE         (TPM_AE)(0x00000000)
1631  typedef struct {
1632       TPM_AT                  tag;
1633       UINT32                 data;
1634  } TPMS_AC_OUTPUT;        // Table 2:220
1635  typedef struct {
1636       UINT32                  count;
1637       TPMS_AC_OUTPUT          acCapabilities[MAX_AC_CAPABILITIES];
1638  } TPML_AC_CAPABILITIES;                   /* Structure */
1639  #endif                      /* _TPM_TYPES_H_ */
5.22 VendorString.h

```c
#ifndef _VENDOR_STRING_H
#define _VENDOR_STRING_H

#define MANUFACTURER    "MSFT"

#ifdef MANUFACTURER
#else
#error MANUFACTURER is not provided. \
Please modify include\VendorString.h to provide a specific \
manufacturer name.
#endif

#define VENDOR_STRING_1       "xCG "
#define VENDOR_STRING_2       "fTPM"
// #define       VENDOR_STRING_3
// #define       VENDOR_STRING_4

#ifndef VENDOR_STRING_1
#else
#error VENDOR_STRING_1 is not provided. \
Please modify include\VendorString.h to provide a vendor specific string.
#endif

#define FIRMWARE_V1         (0x20170619)
#define FIRMWARE_V2         (0x00163636)

#ifndef FIRMWARE_V1
#else
#error FIRMWARE_V1 is not provided. \
Please modify include\VendorString.h to provide a vendor specific firmware \
version
#endif
#endif
```

Define up to 4-byte values for MANUFACTURER. This value defines the response for TPM_PT_MANUFACTURER in TPM2_GetCapability(). The following line should be un-commented and a vendor specific string should be provided here.

Define up to 4, 4-byte values. The values must each be 4 bytes long and the last value used may contain trailing zeros. These values define the response for TPM_PT_VENDOR_STRING_(1-4) in TPM2_GetCapability(). The following line should be un-commented and a vendor specific string should be provided here. The vendor strings 2-4 may also be defined as appropriate.

The more significant 32-bits of a vendor-specific value indicating the version of the firmware. The following line should be un-commented and a vendor specific firmware V1 should be provided here. The FIRMWARE_V2 may also be defined as appropriate.

The less significant 32-bits of a vendor-specific value indicating the version of the firmware.
5.23 swap.h

```c
#ifndef __SWAP_H
#define __SWAP_H

#if LITTLE_ENDIAN_TPM
#define TO_BIG_ENDIAN_UINT16(i) REVERSE_ENDIAN_16(i)
#define FROM_BIG_ENDIAN_UINT16(i) REVERSE_ENDIAN_16(i)
#define TO_BIG_ENDIAN_UINT32(i) REVERSE_ENDIAN_32(i)
#define FROM_BIG_ENDIAN_UINT32(i) REVERSE_ENDIAN_32(i)
#define TO_BIG_ENDIAN_UINT64(i) REVERSE_ENDIAN_64(i)
#define FROM_BIG_ENDIAN_UINT64(i) REVERSE_ENDIAN_64(i)
#else
#define TO_BIG_ENDIAN_UINT16(i) (i)
#define FROM_BIG_ENDIAN_UINT16(i) (i)
#define TO_BIG_ENDIAN_UINT32(i) (i)
#define FROM_BIG_ENDIAN_UINT32(i) (i)
#define TO_BIG_ENDIAN_UINT64(i) (i)
#define FROM_BIG_ENDIAN_UINT64(i) (i)
#endif

#if AUTO_ALIGN == NO
The aggregation macros for machines that do not allow unaligned access or for little-endian machines. Aggregate bytes into a UINT

#define BYTE_ARRAY_TO_UINT8(b)  (uint8_t)((b)[0])
#define BYTE_ARRAY_TO_UINT16(b) ByteArrayToUint16((BYTE *)(b))
#define BYTE_ARRAY_TO_UINT32(b) ByteArrayToUint32((BYTE *)(b))
#define BYTE_ARRAY_TO_UINT64(b) ByteArrayToUint64((BYTE *)(b))
#define UINT8_TO_BYTE_ARRAY(i, b) {((uint8_t  *)(b))[0] = (i);}  
#define UINT16_TO_BYTE_ARRAY(i, b) {((uint16_t *)(b)) = (i);}  
#define UINT32_TO_BYTE_ARRAY(i, b) {((uint32_t *)(b)) = (i);}  
#define UINT64_TO_BYTE_ARRAY(i, b) {((uint64_t *)(b)) = (i);}  
#else
#define BYTE_ARRAY_TO_UINT8(b)  *((uint8_t  *)(b))
#define BYTE_ARRAY_TO_UINT16(b) REVERSE_ENDIAN_16(*((uint16_t *)(b)))
#define BYTE_ARRAY_TO_UINT32(b) REVERSE_ENDIAN_32(*((uint32_t *)(b)))
#define BYTE_ARRAY_TO_UINT64(b) REVERSE_ENDIAN_64(*((uint64_t *)(b)))
#define UINT8_TO_BYTE_ARRAY(i, b) {((uint8_t  *)(b))[0] = (i);}  
#define UINT16_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_16( ((uint16_t *)(b)) = (i);}  
#define UINT32_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_32( ((uint32_t *)(b)) = (i);}  
#define UINT64_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_64( ((uint64_t *)(b)) = (i);}  
#endif
#endif

The aggregation macros for machines that allow unaligned memory access Aggregate a byte array into a UINT

#define BYTE_ARRAY_TO_UINT8(b)  *((uint8_t  *)(b))
#define BYTE_ARRAY_TO_UINT16(b) ByteArrayToUint16((BYTE *)(b))
#define BYTE_ARRAY_TO_UINT32(b) ByteArrayToUint32((BYTE *)(b))
#define BYTE_ARRAY_TO_UINT64(b) ByteArrayToUint64((BYTE *)(b))
#define UINT8_TO_BYTE_ARRAY(i, b) {((uint8_t  *)(b))[0] = (i);}  
#define UINT16_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_16( ((uint16_t *)(b)) = (i);}  
#define UINT32_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_32( ((uint32_t *)(b)) = (i);}  
#define UINT64_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_64( ((uint64_t *)(b)) = (i);}  
#define UINT8_TO_BYTE_ARRAY(i, b) {((uint8_t  *)(b))[0] = (i);}  
#define UINT16_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_16( ((uint16_t *)(b)) = (i);}  
#define UINT32_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_32( ((uint32_t *)(b)) = (i);}  
#define UINT64_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_64( ((uint64_t *)(b)) = (i);}  
```

The big-endian macros for machines that allow unaligned memory access. the big-endian macros for machines that allow unaligned memory access Aggregate a byte array into a UINT

```c
#define BYTE_ARRAY_TO_UINT8(b)  *((uint8_t  *)(b))
#define BYTE_ARRAY_TO_UINT16(b) ByteArrayToUint16((BYTE *)(b))
#define BYTE_ARRAY_TO_UINT32(b) ByteArrayToUint32((BYTE *)(b))
#define BYTE_ARRAY_TO_UINT64(b) ByteArrayToUint64((BYTE *)(b))
#define UINT8_TO_BYTE_ARRAY(i, b) {((uint8_t  *)(b))[0] = (i);}  
#define UINT16_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_16( ((uint16_t *)(b)) = (i);}  
#define UINT32_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_32( ((uint32_t *)(b)) = (i);}  
#define UINT64_TO_BYTE_ARRAY(i, b) {REVERSE_ENDIAN_64( ((uint64_t *)(b)) = (i);}  
```
#define UINT16_TO_BYTE_ARRAY(i, b)  {*((uint16_t *)(b)) = REVERSE_ENDIAN_16(i);}
#define UINT32_TO_BYTE_ARRAY(i, b)  {*((uint32_t *)(b)) = REVERSE_ENDIAN_32(i);}
#define UINT64_TO_BYTE_ARRAY(i, b)  {*((uint64_t *)(b)) = REVERSE_ENDIAN_64(i);}
#endif  // BIG_ENDIAN_TPM
#endif  // AUTO_ALIGN == NO
#endif  // __SWAP_H
6 Main

6.1 Introduction

The files in this section are the main processing blocks for the TPM. ExecuteCommand.c contains the entry point into the TPM code and the parsing of the command header. SessionProcess.c handles the parsing of the session area and the authorization checks, and CommandDispatch.c does the parameter unmarshaling and command dispatch.

6.2 ExecCommand.c

6.2.1 Introduction

This file contains the entry function ExecuteCommand() which provides the main control flow for TPM command execution.

6.2.2 Includes

```c
#include "Tpm.h"
#include "ExecCommand_fp.h"
```

Uncomment this next #include if doing static command/response buffer sizing

```c
// #include "CommandResponseSizes_fp.h"
```

6.2.3 ExecuteCommand()

The function performs the following steps.
a) Parses the command header from input buffer.
b) Calls ParseHandleBuffer() to parse the handle area of the command.
c) Validates that each of the handles references a loaded entity.
d) Calls ParseSessionBuffer() to:
   1) unmarshal and parse the session area;
   2) check the authorizations; and
   3) when necessary, decrypt a parameter.
e) Calls CommandDispatcher() to:
   1) unmarshal the command parameters from the command buffer;
   2) call the routine that performs the command actions; and
   3) marshal the responses into the response buffer.
f) If any error occurs in any of the steps above create the error response and return.
g) Calls BuildResponseSessions() to:
   1) when necessary, encrypt a parameter
   2) build the response authorization sessions
   3) update the audit sessions and nonces
h) Calls BuildResponseHeader() to complete the construction of the response.

responseSize is set by the caller to the maximum number of bytes available in the output buffer. ExecuteCommand() will adjust the value and return the number of bytes placed in the buffer.

response is also set by the caller to indicate the buffer into which ExecuteCommand() is to place the response.

request and response may point to the same buffer

NOTE: As of February, 2016, the failure processing has been moved to the platform-specific code. When the TPM code encounters an unrecoverable failure, it will SET g_inFailureMode and call _plat__Fail(). That function should not return but may call ExecuteCommand().

LIBEXPORT void ExecuteCommand(
    uint32_t requestSize,  // IN: command buffer size
    unsigned char *request, // IN: command buffer
    uint32_t *responseSize, // IN/OUT: response buffer size
    unsigned char **response // IN/OUT: response buffer
)
{
    // Command local variables
    UINT32 commandSize;
    COMMAND command;

    // Response local variables
    UINT32 maxResponse = *responseSize;
    TPM_RC result;          // return code for the command

    // This next function call is used in development to size the command and response
    // buffers. The values printed are the sizes of the internal structures and
    // not the sizes of the canonical forms of the command response structures. Also,
    // the sizes do not include the tag, command.code, requestSize, or the authorization
    // fields.
    //CommandResponseSizes();

    // Set flags for NV access state. This should happen before any other

    // operation that may require a NV write. Note, that this needs to be done
// even when in failure mode. Otherwise, g_updateNV would stay SET while in
// Failure mode and the NV would be written on each call.
g_updateNV = UT_NONE;
g_clearOrderly = FALSE;
if (g_inFailureMode)
{
    // Do failure mode processing
    TpmFailureMode(requestSize, request, responseSize, response);
    return;
}
// Query platform to get the NV state. The result state is saved internally
// and will be reported by NvIsAvailable(). The reference code requires that
// accessibility of NV does not change during the execution of a command.
// Specifically, if NV is available when the command execution starts and then
// is not available later when it is necessary to write to NV, then the TPM
// will go into failure mode.
NvCheckState();

// Due to the limitations of the simulation, TPM clock must be explicitly
// synchronized with the system clock whenever a command is received.
// This function call is not necessary in a hardware TPM. However, taking
// a snapshot of the hardware timer at the beginning of the command allows
// the time value to be consistent for the duration of the command execution.
TimeUpdateToCurrent();

// Any command through this function will unceremoniously end the
// TPM_Hash_Data/_TPM_Hash_End sequence.
if (g_DRTMHandle != TPM_RH_UNASSIGNED)
    ObjectTerminateEvent();

// Get command buffer size and command buffer.
command.parameterBuffer = request;
command.parameterSize = requestSize;

// Parse command header: tag, commandSize and command code.
// First parse the tag. The unmarshaling routine will validate
// that it is either TPM_ST_SESSIONS or TPM_ST_NO_SESSIONS.
result = TPMI_ST_COMMAND_TAG_Unmarshal(&command.tag,
                                        &command.parameterBuffer,
                                        &command.parameterSize);
if (result != TPM_RC_SUCCESS)
    goto Cleanup;

// Unmarshal the commandSize indicator.
result = UINT32_Unmarshal(&commandSize,
                          &command.parameterBuffer,
                          &command.parameterSize);
if (result != TPM_RC_SUCCESS)
    goto Cleanup;

// On a TPM that receives bytes on a port, the number of bytes that were
// received on that port is requestSize it must be identical to commandSize.
// In addition, commandSize must not be larger than MAX_COMMAND_SIZE allowed
// by the implementation. The check against MAX_COMMAND_SIZE may be redundant
// as the input processing (the function that receives the command bytes and
// places them in the input buffer) would likely have the input truncated when
// it reaches MAX_COMMAND_SIZE, and requestSize would not equal commandSize.
if (commandSize != requestSize || commandSize > MAX_COMMAND_SIZE)
{
    result = TPM_RC_COMMAND_SIZE;
    goto Cleanup;
}

// Unmarshal the command code.
result = TPM_CC_Unmarshal(&command.code, &command.parameterBuffer,
                          &command.parameterSize);
if (result != TPM_RC_SUCCESS)
    goto Cleanup;

// Check to see if the command is implemented.
command.index = CommandCodeToCommandIndex(command.code);
if (UNIMPLEMENTED_COMMAND_INDEX == command.index)
{
    result = TPM_RC_COMMAND_CODE;
    goto Cleanup;
}
#endif

if (FIELD_UPGRADE_IMPLEMENTED == YES)
{
    // If the TPM is in FUM, then the only allowed command is
    // TPM_CC_FieldUpgradeData.
    if (IsFieldUpgradeMode() && (command.code != TPM_CC_FieldUpgradeData))
    {
        result = TPM_RC_UPGRADE;
        goto Cleanup;
    }
    else
    { /* EndIf */
        // Excepting FUM, the TPM only accepts TPM2_Startup() after
        // _TPM_Init. After getting a TPM2_Startup(), TPM2_Startup()
        // is no longer allowed.
        if (!TPMIsStarted() && command.code != TPM_CC_Startup)
            result = TPM_RC_INITIALIZE;
        goto Cleanup;
    }
}
// Start regular command process.
NvIndexCacheInit();
// Parse handle buffer.
result = ParseHandleBuffer(&command);
if (result != TPM_RC_SUCCESS)
    goto Cleanup;
// All handles in the handle area are required to reference TPM-resident
// entities.
result = EntityGetLoadStatus(&command);
if (result != TPM_RC_SUCCESS)
    goto Cleanup;
// Authorization session handling for the command.
ClearCpRpHashes(&command);
if (command.tag == TPM_ST_SESSIONS)
{
    // Find out session buffer size.
    result = UINT32_Unmarshal( ( UINT32 * ) &command.authSize,
                               &command.parameterBuffer,
                               &command.parameterSize);
    if (result != TPM_RC_SUCCESS)
        goto Cleanup;
    // Perform sanity check on the unmarshaled value. If it is smaller than
    // the smallest possible session or larger than the remaining size of
    // the command, then it is an error. NOTE: This check could pass but the
    // session size could still be wrong. That will be determined after the
    // sessions are unmarshaled.
    if (command.authSize < 9 || command.authSize > command.parameterSize)
    {
        result = TPM_RC_SIZE;
        goto Cleanup;
    }
    command.parameterSize -= command.authSize;
}
// The actions of ParseSessionBuffer() are described in the introduction.
// As the sessions are parsed command.parameterBuffer is advanced so, on a
// successful return, command.parameterBuffer should be pointing at the
// first byte of the parameters
result = ParseSessionBuffer(&command);
if (result != TPM_RC_SUCCESS)
    goto Cleanup;
Part 4: Supporting Routines

Trusted Platform Module Library

Family “2.0”

TCG Public Review

Page 185

160 }
161 else
162 {
163     command.authSize = 0;
164     // The command has no authorization sessions.
165     // If the command requires authorizations, then CheckAuthNoSession() will
166     // return an error.
167     result = CheckAuthNoSession(&command);
168     if(result != TPM_RC_SUCCESS)
169         goto Cleanup;
170 }
171
172 // Set up the response buffer pointers. CommandDispatch will marshal the
173 // response parameters starting at the address in command.responseBuffer.
174 // *response = MemoryGetResponseBuffer(command.index);
175 // leave space for the command header
176 command.responseBuffer = *response + STD_RESPONSE_HEADER;
177
178 // leave space for the parameter size field if needed
179 if(command.tag == TPM_ST_SESSIONS)
180     command.responseBuffer += sizeof(UINT32);
181 if(IsHandleInResponse(command.index))
182     command.responseBuffer += sizeof(TPM_HANDLE);
183
184 // CommandDispatcher returns a response handle buffer and a response parameter
185 // buffer if it succeeds. It will also set the parameterSize field in the
186 // buffer if the tag is TPM RC_SESSIONS.
187 result = CommandDispatcher(&command);
188 if(result != TPM_RC_SUCCESS)
189     goto Cleanup;
190
191 // Build the session area at the end of the parameter area.
192 BuildResponseSession(&command);
193
194 Cleanup:
195     if(g_clearOrderly == TRUE
196         && NV_IS_ORDERLY)
197         {
198         #if USE_DA_USED
199             gp.orderlyState = g_daUsed ? SU_DA_USED_VALUE : SU_NONE_VALUE;
200         #else
201             gp.orderlyState = SU_NONE_VALUE;
202         #endif
203             NV_SYNC_PERSISTENT(orderlyState);
204         }
205         // This implementation loads an "evict" object to a transient object slot in
206         // RAM whenever an "evict" object handle is used in a command so that the
207         // access to any object is the same. These temporary objects need to be
208         // cleared from RAM whether the command succeeds or fails.
209         ObjectCleanupEvict();
210
211         // The parameters and sessions have been marshaled. Now tack on the header and
212         // set the sizes
213         BuildResponseHeader(&command, *response, result);
214
215         // Try to commit all the writes to NV if any NV write happened during this
216         // command execution. This check should be made for both succeeded and failed
217         // commands, because a failed one may trigger a NV write in DA logic as well.
218         // This is the only place in the command execution path that may call the NV
219         // commit. If the NV commit fails, the TPM should be put in failure mode.
220         if((g_updateNV != UT_NONE) && !g_inFailureMode)
221             {
222                 if(g_updateNV == UT_ORDERLY)
223                     NvUpdateIndexOrderlyData();
224                 if(!NvCommit())
225                     FAIL(FATAL_ERROR_INTERNAL);
226                 g_updateNV = UT_NONE;
pAssert((UINT32)command.parameterSize <= maxResponse);

// Clear unused bits in response buffer.
MemorySet(*response + *responseSize, 0, maxResponse - *responseSize);

// as a final act, and not before, update the response size.
*responseSize = (UINT32)command.parameterSize;

return;
6.3 CommandDispatcher.c

6.3.1 Introduction

CommandDispatcher() performs the following operations:

- unmarshals command parameters from the input buffer;

**NOTE 1** Unlike other unmarshaling functions, parmBufferStart does not advance. parmBufferSize is reduced.

- invokes the function that performs the command actions;
- marshals the returned handles, if any; and
- marshals the returned parameters, if any, into the output buffer putting in the parameterSize field if authorization sessions are present.

**NOTE 2** The output buffer is the return from the MemoryGetResponseBuffer() function. It includes the header, handles, response parameters, and authorization area. respParmSize is the response parameter size, and does not include the header, handles, or authorization area.

**NOTE 3** The reference implementation is permitted to do compare operations over a union as a byte array. Therefore, the command parameter in structure must be initialized (e.g., zeroed) before unmarshaling so that the compare operation is valid in cases where some bytes are unused.

6.3.1.1 Includes and Typedefs

```c
#include "Tpm.h"

#define TABLE_DRIVEN_DISPATCH

typedef TPM_RC(NoFlagFunction)(void *target, BYTE **buffer, INT32 *size);
typedef TPM_RC(FlagFunction)(void *target, BYTE **buffer, INT32 *size, BOOL flag);
typedef INT16(MarshalFunction)(void *source, BYTE **buffer, INT32 *size);
typedef MarshalFunction *MARSHAL_t;
typedef TPM_RC(COMMAND_NO_ARGS)(void);
typedef TPM_RC(COMMAND_IN_ARG)(void *in);
typedef TPM_RC(COMMAND_OUT_ARG)(void *out);
typedef TPM_RC(COMMAND_INOUT_ARG)(void *in, void *out);
typedef union
{
    COMMAND_NO_ARGS *noArgs;
    COMMAND_IN_ARG *inArg;
    COMMAND_OUT_ARG *outArg;
    COMMAND_INOUT_ARG *inOutArg;
} COMMAND_t;
```

This structure is used by ParseHandleBuffer() and CommandDispatcher(). The parameters in this structure are unique for each command. The parameters are:
<table>
<thead>
<tr>
<th><strong>command</strong></th>
<th>Holds the address of the command processing function that is called by Command Dispatcher.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>inSize</strong></td>
<td>This is the size of the command-dependent input structure. The input structure holds the unmarshaled handles and command parameters. If the command takes no arguments (handles or parameters) then inSize will have a value of 0.</td>
</tr>
<tr>
<td><strong>outSize</strong></td>
<td>This is the size of the command-dependent output structure. The output structure holds the results of the command in an unmarshaled form. When command processing is completed, these values are marshaled into the output buffer. It is always the case that the unmarshaled version of an output structure is larger than the marshaled version. This is because the marshaled version contains the exact same number of significant bytes but with padding removed. typesOffsets -- This parameter points to the list of data types that are to be marshaled or unmarshaled. The list of types follows the offsets array. The offsets array is variable sized so the typesOffset field is necessary for the handle and command processing to be able to find the types that are being handled. The offsets array may be empty. The types structure is described below.</td>
</tr>
<tr>
<td><strong>offsets</strong></td>
<td>This is an array of offsets of each of the parameters in the command or response. When processing the command parameters (not handles) the list contains the offset of the next parameter. For example, if the first command parameter has a size of 4 and there is a second command parameter, then the offset would be 4, indicating that the second parameter starts at 4. If the second parameter has a size of 8, and there is a third parameter, then the second entry in offsets is 12 (4 for the first parameter and 8 for the second). An offset value of 0 in the list indicates the start of the response parameter list. When CommandDispatcher() hits this value, it will stop unmarshaling the parameters and call 'command.' If a command has no response parameters and only one command parameter, then offsets can be an empty list.</td>
</tr>
</tbody>
</table>

```c
typedef struct
{
    COMMAND_t command; // Address of the command
    UINT16 inSize;     // Maximum size of the input structure
    UINT16 outSize;    // Maximum size of the output structure
    UINT16 typesOffset; // address of the types field
    UINT16 offsets[1];
} COMMAND_DESCRIPTOR_t;
```

The types list is an encoded byte array. The byte value has two parts. The most significant bit is used when a parameter takes a flag and indicates if the flag should be SET or not. The remaining 7 bits are an index into an array of addresses of marshaling and unmarshaling functions. The array of functions is divided into 6 sections with a value assigned to denote the start of that section (and the end of the previous section). The defined offset values for each section are:
0 unmarshaling for handles that do not take flags

HANDLE_FIRST_FLAG_TYPE unmarshaling for handles that take flags

PARAMETER_FIRST_TYPE unmarshaling for parameters that do not take flags

PARAMETER_FIRST_FLAG_TYPE unmarshaling for parameters that take flags

PARAMETER_LAST_TYPE + 1 marshaling for handles

RESPONSE_PARAMETER_FIRST_TYPE marshaling for parameters

RESPONSE_PARAMETER_LAST_TYPE is the last value in the list of marshaling and unmarshaling functions

The types list is constructed with a byte of 0xff at the end of the command parameters and with an 0xff at the end of the response parameters.

6.3.1.2 Marshal/Unmarshal Functions

6.3.1.2.1 ParseHandleBuffer()

This is the table-driven version of the handle buffer unmarshaling code

```
TPM_RC
ParseHandleBuffer( *command )
{
    TPM_RC result;
#if TABLE_DRIVEN_DISPATCH
    COMMAND_DESCRIPTOR_t *desc;
    BYTE *types;
    BYTE type;
    BYTE dtype;
#endif
    // Make sure that nothing strange has happened
    pAssert(command->index < sizeof(s_CommandDataArray) / sizeof(COMMAND_DESCRIPTOR_t *));
    // Get the address of the descriptor for this command
    desc = s_CommandDataArray[command->index];
    pAssert(desc != NULL);
    // Get the associated list of unmarshaling data types.
    types = &((BYTE *)desc)[desc->typesOffset];
    // if(s_CCAttr[commandIndex].commandIndex == TEST_COMMAND)
    //     commandIndex = commandIndex;
    // No handles yet
    command->handleNum = 0;
```
65  // Get the first type value
66  for(type = *types++;
67     // check each byte to make sure that we have not hit the start
68     // of the parameters
69     (dtype = (type & 0x7F)) < PARAMETER_FIRST_TYPE;
70     // get the next type
71     type = *types++)
72   {
73     // See if unmarshaling of this handle type requires a flag
74     if(dtype < HANDLE_FIRST_FLAG_TYPE)
75     {
76       // Look up the function to do the unmarshaling
77       NoFlagFunction *f = (NoFlagFunction *)UnmarshalArray[dtype];
78       // call it
79       result = f(&command->handles[command->handleNum]),
80       &command->parameterBuffer,
81       &command->parameterSize);  
82     }
83     else
84     {
85       // Look up the function
86       FlagFunction *f = UnmarshalArray[dtype];
87       // Call it setting the flag to the appropriate value
88       result = f(&command->handles[command->handleNum]),
89       &command->parameterBuffer,
90       &command->parameterSize, (type & 0x80) != 0);  
91     }
92     // Got a handle
93     // We do this first so that the match for the handle offset of the
94     // response code works correctly.
95     command->handleNum += 1;
96     if(result != TPM_RC_SUCCESS)
97     // if the unmarshaling failed, return the response code with the
98     // handle indication set
99     return result + TPM_RC_H + (command->handleNum * TPM_RC_1);
100   }
101   #else
102     BYTE            **handleBufferStart = &command->parameterBuffer;
103     INT32           *bufferRemainingSize = &command->parameterSize;
104     TPM_HANDLE      *handles = &command->handles[0];
105     UINT32          *handleCount = &command->handleNum;
106     *handleCount = 0;
107     switch(command->code)
108     {
109       #include "HandleProcess.h"
110       #undef handles
111       default:
112         FAIL(FATAL_ERROR_INTERNAL);
113         break;
114       #endif
115     return TPM_RC_SUCCESS;
116   }
117   #endif

6.3.1.2.2 CommandDispatcher()

Function to unmarshal the command parameters, call the selected action code, and marshal the
response parameters.

TPM_RC
CommandDispatcher(
    COMMAND *command


```c
123 }
124 {
125 #if !TABLE_DRIVEN_DISPATCH
126 TPM_RC result;
127 BYTE **paramBuffer = &command->parameterBuffer;
128 INT32 *paramBufferSize = &command->parameterSize;
129 BYTE **responseBuffer = &command->responseBuffer;
130 INT32 *respParmSize = &command->parameterSize;
131 INT32 rSize;
132 TPM_HANDLE *handles = &command->handles[0];
133
134 // command->handleNum = 0; // The command-specific code knows how
135 // many handles there are. This is for
136 // cataloging the number of response
137 // handles
138 MemoryIoBufferAllocationReset(); // Initialize so that allocation will
139 // work properly
140 switch(GetCommandCode(command->index))
141 {
142 #include "CommandDispatcher.h"
143
default:
144     FAIL(FATAL_ERROR_INTERNAL);
145     break;
146 }
147 Exit:
148     MemoryIoBufferZero();
149     return result;
150 #else
151
152 COMMAND_DESCRIPTOR_t *desc;
153 BYTE *types;
154 BYTE type;
155 UINT16 *offsets;
156 UINT16 offset = 0;
157 UINT32 maxInSize;
158 BYTE *commandIn;
159 INT32 maxOutSize;
160 BYTE *commandOut;
161 COMMAND_t cmd;
162 TPM_HANDLE *handles;
163 UINT32 hasInParameters = 0;
164 BOOL hasOutParameters = FALSE;
165 UINT32 pNum = 0;
166 BYTE dType; // dispatch type
167 TPM_RC result;
168
169 // Get the address of the descriptor for this command
170 pAssert(command->index < sizeof(s_CommandDataArray) / sizeof(COMMAND_DESCRIPTOR_t *));
171 desc = s_CommandDataArray[command->index];
172
173 // Get the list of parameter types for this command
174 pAssert(desc != NULL);
175 types = &((BYTE *)desc)[desc->typesOffset];
176
177 // Get a pointer to the list of parameter offsets
178 offsets = &desc->offsets[0];
179 // pointer to handles
180 handles = command->handles;
181
182 // Get the size required to hold all the unmarshaled parameters for this command
183 maxInSize = desc->inSize;
184 // and the size of the output parameter structure returned by this command
185 maxOutSize = desc->outSize;
186
187 MemoryIoBufferAllocationReset();
```
// Get a buffer for the input parameters
commandIn = MemoryGetInBuffer(maxInSize);
// And the output parameters
commandOut = (BYTE *)MemoryGetOutBuffer((UINT32)maxOutSize);

// Get the address of the action code dispatch
cmd = desc->command;

// Copy any handles into the input buffer
for (type = *types++; (type & 0x7F) < PARAMETER_FIRST_TYPE; type = *types++)
{
    // 'offset' was initialized to zero so the first unmarshaling will always
    // be to the start of the data structure
    *(TPM_HANDLE *)&(commandIn[offset]) = *handles++;
    // This check is used so that we don't have to add an additional offset
    // value to the offsets list to correspond to the stop value in the
    // command parameter list.
    if (*types != 0xFF)
        offset = *offsets++;
    // maxInSize -= sizeof(TPM_HANDLE);
    hasInParameters++;
}
// Exit loop with type containing the last value read from types
// maxInSize has the amount of space remaining in the command action input
// buffer. Make sure that we don't have more data to unmarshal than is going to
// fit.

// type contains the last value read from types so it is not necessary to
// reload it, which is good because *types now points to the next value
for (; (dType = (type & 0x7F)) <= PARAMETER_LAST_TYPE; type = *types++)
{
    pNum++;
    if (dType < PARAMETER_FIRST_FLAG_TYPE)
    {
        NoFlagFunction *f = (NoFlagFunction *)UnmarshalArray[dType];
        result = f(&commandIn[offset], &command->parameterBuffer,
                   &command->parameterSize);
    }
    else
    {
        FlagFunction   *f = UnmarshalArray[dType];
        result = f(&commandIn[offset], &command->parameterBuffer,
                   &command->parameterSize,
                   (type & 0x80) != 0);
    }
    if (result != TPM_RC_SUCCESS)
    {
        result += TPM_RC_P + (TPM_RC_1 * pNum);
        goto Exit;
    }
    // This check is used so that we don't have to add an additional offset
    // value to the offsets list to correspond to the stop value in the
    // command parameter list.
    if (*types != 0xFF)
    {
        offset = *offsets++;
        hasInParameters++;
    }
    // Should have used all the bytes in the input
    if (command->parameterSize != 0)
    {
        result = TPM_RC_SIZE;
        goto Exit;
    }
}
// The command parameter unmarshaling stopped when it hit a value that was out
// of range for unmarshaling values and left *types pointing to the first
// marshaling type. If that type happens to be the STOP value, then there
// are no response parameters. So, set the flag to indicate if there are
// output parameters.
hasOutParameters = *types != 0xFF;

// There are four cases for calling, with and without input parameters and with
// and without output parameters.
if(hasInParameters > 0)
{
    if(hasOutParameters)
        result = cmd.inOutArg(commandIn, commandOut);
    else
        result = cmd.inArg(commandIn);
}
else
{
    if(hasOutParameters)
        result = cmd.outArg(commandOut);
    else
        result = cmd.noArgs();
}
if(result != TPM_RC_SUCCESS)
    goto Exit;

// Offset in the marshaled output structure
offset = 0;

// Process the return handles, if any
command->handleNum = 0;

// Could make this a loop to process output handles but there is only ever
// one handle in the outputs (for now).
type = *types++;
if((dType = (type & 0x7F)) < RESPONSE_PARAMETER_FIRST_TYPE)
{
    // The out->handle value was referenced as TPM_HANDLE in the
    // action code so it has to be properly aligned.
    command->handles[command->handleNum++] =
        *((TPM_HANDLE *)&(commandOut[offset]));
    maxOutSize -= sizeof(UINT32);
    type = *types++;
    offset = *offsets++;
}
// Use the size of the command action output buffer as the maximum for the
// number of bytes that can get marshaled. Since the marshaling code has
// no pointers to data, all of the data being returned has to be in the
// command action output buffer. If we try to marshal more bytes than
// could fit into the output buffer, we need to fail.
for(;(dType = (type & 0x7F)) <= RESPONSE_PARAMETER_LAST_TYPE
    && !_inFailureMode; type = *types++)
{
    const MARSHAL_t     f = MarshalArray[dType];

    command->parameterSize += f(&commandOut[offset], &command->responseBuffer,
        &maxOutSize);
    offset = *offsets++;
}
result = (maxOutSize < 0) ? TPM_RC_FAILURE : TPM_RC_SUCCESS;
Exit:
MemoryIoBufferZero();
return result;
#endif
6.4 SessionProcess.c

6.4.1 Introduction

This file contains the subsystem that process the authorization sessions including implementation of the Dictionary Attack logic. ExecCommand() uses ParseSessionBuffer() to process the authorization session area of a command and BuildResponseSession() to create the authorization session area of a response.

6.4.2 Includes and Data Definitions

```c
#define SESSION_PROCESS_C
#include "Tpm.h"
```

6.4.3 Authorization Support Functions

6.4.3.1 IsDAExempted()

This function indicates if a handle is exempted from DA logic. A handle is exempted if it is

a) a primary seed handle,

b) an object with noDA bit SET,

c) an NV Index with TPMA_NV_NO_DA bit SET, or

d) a PCR handle.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>handle is exempted from DA logic</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>handle is not exempted from DA logic</td>
</tr>
</tbody>
</table>

```c
BOOL IsDAExempted(TPM_HANDLE handle) // IN: entity handle
{
    BOOL result = FALSE;
    //
    switch(HandleGetType(handle))
    {
        case TPM_HT_PERMANENT:
        // All permanent handles, other than TPM_RH_LOCKOUT, are exempt from
        // DA protection.
        result = (handle != TPM_RH_LOCKOUT);
        break;
        // When this function is called, a persistent object will have been loaded
        // into an object slot and assigned a transient handle.
        case TPM_HT_TRANSIENT:
        {
            TPMA_OBJECT attributes = ObjectGetPublicAttributes(handle);
            result = IS_ATTRIBUTE(attributes, TPMA_OBJECT, noDA);
            break;
        }
        case TPM_HT_NV_INDEX:
        {
            NV_INDEX *nvIndex = NvGetIndexInfo(handle, NULL);
            result = IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, NO_DA);
            break;
        }
    }
    return result;
}
```
case TPM_HT_PCR:
    // PCRs are always exempted from DA.
    result = TRUE;
    break;
default:
    break;
}
return result;
}

6.4.3.2 IncrementLockout()

This function is called after an authorization failure that involves use of an authValue. If the entity referenced by the handle is not exempt from DA protection, then the failedTries counter will be incremented.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_AUTH_FAIL</td>
<td>authorization failure that caused DA lockout to increment</td>
</tr>
<tr>
<td>TPM_RC_BAD_AUTH</td>
<td>authorization failure did not cause DA lockout to increment</td>
</tr>
</tbody>
</table>

```
static TPM_RC IncrementLockout(
    UINT32           sessionIndex
)
{
    TPM_HANDLE       handle = s_associatedHandles[sessionIndex];
    TPM_HANDLE       sessionHandle = s_sessionHandles[sessionIndex];
    SESSION         *session = NULL;
    // Don't increment lockout unless the handle associated with the session
    // is DA protected or the session is bound to a DA protected entity.
    if(sessionHandle == TPM_RS_PW)
    {
        if(IsDAExempted(handle))
            return TPM_RC_BAD_AUTH;
    }
    else
    {
        session = SessionGet(sessionHandle);
        // If the session is bound to lockout, then use that as the relevant
        // handle. This means that an authorization failure with a bound session
        // bound to lockoutAuth will take precedence over any other
        // lockout check
        if(session->attributes.isLockoutBound == SET)
            handle = TPM_RH_LOCKOUT;
        if(session->attributes.isDaBound == CLEAR
            && (IsDAExempted(handle) || session->attributes.includeAuth == CLEAR))
            // If the handle was changed to TPM_RH_LOCKOUT, this will not return
            // TPM_RC_BAD_AUTH
            return TPM_RC_BAD_AUTH;
    }
    if(handle == TPM_RH_LOCKOUT)
    {
        pAssert(gp.lockOutAuthEnabled == TRUE);
        // lockout is no longer enabled
        gp.lockOutAuthEnabled = FALSE;
        // For TPM_RH_LOCKOUT, if lockoutRecovery is 0, no need to update NV since
        // the lockout authorization will be reset at startup.
        if(gp.lockoutRecovery != 0)
            
```
82     if(NV_IS_AVAILABLE)
83         // Update NV.
84         NV_SYNC_PERSISTENT(lockOutAuthEnabled);
85     else
86         // No NV access for now. Put the TPM in pending mode.
87         s_DAPendingOnNV = TRUE;
88 }
89 else
90 {
91     if(gp.recoveryTime != 0)
92     {
93         gp.failedTries++;
94         if(NV_IS_AVAILABLE)
95             // Record changes to NV. NvWrite will SET g_updateNV
96             NV_SYNC_PERSISTENT(failedTries);
97     else
98         // No NV access for now. Put the TPM in pending mode.
99         s_DAPendingOnNV = TRUE;
100     }
101     // Register a DA failure and reset the timers.
102     DARegisterFailure(handle);
103     return TPM_RC_AUTH_FAIL;
104 }

6.4.3.3 IsSessionBindEntity()

This function indicates if the entity associated with the handle is the entity, to which this session is bound. The binding would occur by making the bind parameter in TPM2_StartAuthSession() not equal to TPM_RH_NULL. The binding only occurs if the session is an HMAC session. The bind value is a combination of the Name and the authValue of the entity.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>handle points to the session start entity</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>handle does not point to the session start entity</td>
</tr>
</tbody>
</table>

108 static BOOL IsSessionBindEntity(
109     TPM_HANDLE associatedHandle, // IN: handle to be authorized
110     SESSION *session // IN: associated session
111 )
112 {
113     TPM2B_NAME entity; // The bind value for the entity
114     // If the session is not bound, return FALSE.
115     if(session->attributes.isBound)
116     {
117         // Compute the bind value for the entity.
118         SessionComputeBoundEntity(associatedHandle, &entity);
119         // Compare to the bind value in the session.
120         return MemoryEqual2B(&entity.b, &session->ul.boundEntity.b);
121     }
122     return FALSE;
123 }

6.4.3.4 `IsPolicySessionRequired()`

Checks if a policy session is required for a command. If a command requires DUP or ADMIN role authorization, then the handle that requires that role is the first handle in the command. This simplifies this checking. If a new command is created that requires multiple ADMIN role authorizations, then it will have to be special-cased in this function. A policy session is required if:

a) the command requires the DUP role,

b) the command requires the ADMIN role and the authorized entity is an object and its `adminWithPolicy` bit is SET, or

c) the command requires the ADMIN role and the authorized entity is a permanent handle or an NV Index.

d) The authorized entity is a PCR belonging to a policy group, and has its policy initialized

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>policy session is required</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>policy session is not required</td>
</tr>
</tbody>
</table>

```
static BOOL
IsPolicySessionRequired(
    COMMAND_INDEX    commandIndex, // IN: command index
    UINT32           sessionIndex  // IN: session index
)
{
    // We allow an exception for ADMIN role in a transient object. If the object
    // allows ADMIN role actions with authorization, then policy is not
    // required. For all other cases, there is no way to override the command
    // requirement that a policy be used

    if(type == TPM_HT_TRANSIENT)
    {
        OBJECT      *object = HandleToObject(s_associatedHandles[sessionIndex]);

        if(!IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT,
                        adminWithPolicy))
            return FALSE;
    }

    return TRUE;
}
```
6.4.3.5 IsAuthValueAvailable()

This function indicates if authValue is available and allowed for USER role authorization of an entity.

This function is similar to IsAuthPolicyAvailable() except that it does not check the size of the authValue as IsAuthPolicyAvailable() does (a null authValue is a valid authorization, but a null policy is not a valid policy).

This function does not check that the handle reference is valid or if the entity is in an enabled hierarchy. Those checks are assumed to have been performed during the handle unmarshaling.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>authValue is available</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>authValue is not available</td>
</tr>
</tbody>
</table>

```c
static BOOL IsAuthValueAvailable(
    TPM_HANDLE       handle,       // IN: handle of entity
    COMMAND_INDEX    commandIndex, // IN: command index
    UINT32           sessionIndex  // IN: session index
) {
    BOOL             result = FALSE;
    //
    switch(HandleGetType(handle)) {
    case TPM_HT_PERMANENT:
        switch(handle) {
            // At this point hierarchy availability has already been
            // checked so primary seed handles are always available here
            case TPM_RH_OWNER:
            case TPM_RH_ENDORSEMENT:
            case TPM_RH_PLATFORM:
                #ifdef VENDOR_PERMANENT
                    // This vendor defined handle associated with the
                    // manufacturer's shared secret
                case VENDOR_PERMANENT:
                    #endif
                    // The DA checking has been performed on LockoutAuth but we
                    // bypass the DA logic if we are using lockout policy. The
                    // policy would allow execution to continue an lockoutAuth
                    // could be used, even if direct use of lockoutAuth is disabled
                    case TPM_RH_LOCKOUT:
                        // NullAuth is always available.
                case TPM_RH_NULL:
                    result = TRUE;
                    break;
            default:
                // Otherwise authValue is not available.
                    break;
        }
    case TPM_HT_TRANSIENT:
        // A persistent object has already been loaded and the internal
        // handle changed.
        {
            OBJECT       *object;
            TPMA_OBJECT   attributes;
            //
            object = HandleToObject(handle);
            attributes = object->publicArea.objectAttributes;
```

```
```
217 // authValue is always available for a sequence object.
218 // An alternative for this is to
219 // SET_ATTRIBUTE(object->publicArea, TPMA_OBJECT, userWithAuth) when the
220 // sequence is started.
221 if(ObjectIsSequence(object))
222 {
223     result = TRUE;
224     break;
225 }
226 // authValue is available for an object if it has its sensitive
227 // portion loaded and
228 // 1. userWithAuth bit is SET, or
229 // 2. ADMIN role is required
230 if(object->attributes.publicOnly == CLEAR
231     && (IS_ATTRIBUTE(attributes, TPMA_OBJECT, userWithAuth)
232         || (CommandAuthRole(commandIndex, sessionIndex) == AUTH_ADMIN
233             && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, adminWithPolicy))))
234     result = TRUE;
235 }
236 break;
237 case TPM_HT_NV_INDEX:
238 // NV Index.
239 {
240     NV_REF      locator;
241     NV_INDEX    *nvIndex = NvGetIndexInfo(handle, &locator);
242     TPMA_NV     nvAttributes;
243     // pAssert(nvIndex != 0);
244     nvAttributes = nvIndex->publicArea.attributes;
245     if(IsWriteOperation(commandIndex))
246     {
247         // AuthWrite can't be set for a PIN index
248         if(IS_ATTRIBUTE(nvAttributes, TPMA_NV, AUTHWRITE))
249             result = TRUE;
250     }
251     else
252     {
253         // A "read" operation.
254         // For a PIN Index, the authValue is available as long as the
255         // Index has been written and the pinCount is less than pinLimit
256         if(IsNvPinFailIndex(nvAttributes)
257             || IsNvPinPassIndex(nvAttributes))
258             result = TRUE;
259     }
260     else
261     {
262         // For non-PIN Indexes, need to allow use of the authValue
263         if(IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITTEN))
264             break; // return false
265     // get the index values
266     pin.intVal = NvGetUINT64Data(nvIndex, locator);
267     if(pin.pin.pinCount < pin.pin.pinLimit)
268         result = TRUE;
269     }
270     // For non-PIN Indexes, need to allow use of the authValue
271     else if(IS_ATTRIBUTE(nvAttributes, TPMA_NV, AUTHREAD))
272         result = TRUE;
273 }
274 break;
275 case TPM_HT_PCR:
276 // PCR handle.
277 // authValue is always allowed for PCR
278 result = TRUE;
279 break;
280 default:
281 // Otherwise, authValue is not available
282 {
6.4.3.6 IsAuthPolicyAvailable()

This function indicates if an authPolicy is available and allowed.

This function does not check that the handle reference is valid or if the entity is in an enabled hierarchy. Those checks are assumed to have been performed during the handle unmarshaling.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>authPolicy is available</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>authPolicy is not available</td>
</tr>
</tbody>
</table>

```c
static BOOL IsAuthPolicyAvailable(
    TPM_HANDLE handle, // IN: handle of entity
    COMMAND_INDEX commandIndex, // IN: command index
    UINT32 sessionIndex // IN: session index
)
{
    BOOL result = FALSE;

    // switch(HandleGetGetType(handle))
    {
        case TPM_HT_PERMANENT:
            switch(handle)
            {
                // At this point hierarchy availability has already been checked.
                case TPM_RH_OWNER:
                    if(gp.ownerPolicy.t.size != 0)
                    {
                        result = TRUE;
                        break;
                    }
                    case TPM_RH_ENDORSEMENT:
                        if(gp.endorsementPolicy.t.size != 0)
                        {
                            result = TRUE;
                            break;
                        }
                        case TPM_RH_PLATFORM:
                            if(gc.platformPolicy.t.size != 0)
                            {
                                result = TRUE;
                                break;
                            }
                            case TPM_RH_LOCKOUT:
                                if(gp.lockoutPolicy.t.size != 0)
                                {
                                    result = TRUE;
                                    break;
                                }
                                default:
                                {
                                    break;
                                }
                            case TPM_HT_TRANSIENT:
                            {
                                // Object handle.
                                // An evict object would already have been loaded and given a
                                // transient object handle by this point.
                                OBJECT *object = HandleToObject(handle);
                                // Policy authorization is not available for an object with only
                                // public portion loaded.
                                if(object->attributes.publicOnly == CLEAR)
                                {
                                    // Policy authorization is always available for an object but
```
333     // is never available for a sequence.
334     if (!ObjectIsSequence(object))
335         result = TRUE;
336     break;
337 }
338 }
339 case TPM_HT_NV_INDEX:
340     // An NV Index.
341 {
342     NV_INDEX         *nvIndex = NvGetIndexInfo(handle, NULL);
343     TPMA_NV           nvAttributes = nvIndex->publicArea.attributes;
344     // If the policy size is not zero, check if policy can be used.
345     if (nvIndex->publicArea.authPolicy.t.size != 0)
346     {
347         // If policy session is required for this handle, always
348         // uses policy regardless of the attributes bit setting
349         if (IsPolicySessionRequired(commandIndex, sessionIndex))
350             result = TRUE;
351         // Otherwise, the presence of the policy depends on the NV
352         // attributes
353         else if (IsWriteOperation(commandIndex))
354             {
355                 if (IS_ATTRIBUTE(nvAttributes, TPMA_NV, POLICYWRITE))
356                 result = TRUE;
357             }
358         else
359             {
360                 if (IS_ATTRIBUTE(nvAttributes, TPMA_NV, POLICYREAD))
361                 result = TRUE;
362             }
363     }
364     break;
365 }
366 case TPM_HT_PCR:
367     // PCR handle.
368     if (PCRPolicyIsAvailable(handle))
369         result = TRUE;
370     break;
371 default:
372     break;
373 }
374 return result;
375 }

6.4.4 Session Parsing Functions

6.4.4.1 ClearCpRpHashes()

void ClearCpRpHashes(
    COMMAND  *command
)
{
#if ALG_SHA1
    command->sha1CpHash.t.size = 0;
    command->sha1RpHash.t.size = 0;
#endif
#if ALG_SHA256
    command->sha256CpHash.t.size = 0;
    command->sha256RpHash.t.size = 0;
#endif
#if ALG_SHA384
    command->sha384CpHash.t.size = 0;
#endif
command->sha384RpHash.t.size = 0;
#endif
#if ALG_SHA512
command->sha512CpHash.t.size = 0;
command->sha512RpHash.t.size = 0;
#endif
#if ALG_SHA3_256
command->sm3_256CpHash.t.size = 0;
command->sm3_256RpHash.t.size = 0;
#endif
}

6.4.4.2 GetCpHashPointer()

Function to get a pointer to the cpHash of the command

static TPM2B_DIGEST *
GetCpHashPointer(
COMMAND         *command,
TPMI_ALG_HASH    hashAlg
)
{
switch (hashAlg)
#if ALG_SHA1
  case ALG_SHA1_VALUE:
    return (TPM2B_DIGEST *)&command->sha1CpHash;
#endif
#if ALG_SHA256
  case ALG_SHA256_VALUE:
    return (TPM2B_DIGEST *)&command->sha256CpHash;
#endif
#if ALG_SHA384
  case ALG_SHA384_VALUE:
    return (TPM2B_DIGEST *)&command->sha384CpHash;
#endif
#if ALG_SHA512
  case ALG_SHA512_VALUE:
    return (TPM2B_DIGEST *)&command->sha512CpHash;
#endif
#if ALG_SM3_256
  case ALG_SM3_256_VALUE:
    return (TPM2B_DIGEST *)&command->sm3_256CpHash;
#endif
  default:
    break;
return NULL;
}

6.4.4.3 GetRpHashPointer()

Function to get a pointer to the RpHash() of the command

static TPM2B_DIGEST *
GetRpHashPointer(
COMMAND         *command,
TPMI_ALG_HASH    hashAlg
)
{
switch (hashAlg)
#if ALG_SHA1

case ALG_SHA1_VALUE:
    return (TPM2B_DIGEST *)&command->sha1RpHash;
#endif
#if ALG_SHA256
    case ALG_SHA256_VALUE:
        return (TPM2B_DIGEST *)&command->sha256RpHash;
#endif
#if ALG_SHA384
    case ALG_SHA384_VALUE:
        return (TPM2B_DIGEST *)&command->sha384RpHash;
#endif
#if ALG_SHA512
    case ALG_SHA512_VALUE:
        return (TPM2B_DIGEST *)&command->sha512RpHash;
#endif
#if ALG_SM3_256
    case ALG_SM3_256_VALUE:
        return (TPM2B_DIGEST *)&command->sm3_256RpHash;
#endif
default:
    break;
}
return NULL;
}

6.4.4.4 ComputeCpHash()

This function computes the cpHash as defined in Part 2 and described in Part 1.

static TPM2B_DIGEST * ComputeCpHash(
    COMMAND *command,           // IN: command parsing structure
    TPMI_ALG_HASH hashAlg       // IN: hash algorithm
) {
    UINT32               i;
    HASH_STATE           hashState;
    TPM2B_NAME           name;
    TPM2B_DIGEST        *cpHash;
// cpHash = hash(commandCode [ || authName1
// [ || authName2
// [ || authName 3 ]]]
// [ || parameters])
// A cpHash can contain just a commandCode only if the lone session is
// an audit session.
// Get pointer to the hash value
    cpHash = GetCpHashPointer(command, hashAlg);
    if(cpHash->t.size == 0) {
        cpHash->t.size = CryptHashStart(&hashState, hashAlg);
        // Add commandCode.
        CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), command->code);
        // Add authNames for each of the handles.
        for(i = 0; i < command->handleNum; i++)
            CryptDigestUpdate2B(&hashState, &EntityGetName(command->handles[i],
                                 &name)->b);
        // Add the parameters.
        CryptDigestUpdate(&hashState, command->parameterSize,
                          command->parameterBuffer);
        // Complete the hash.
        CryptHashEnd2B(&hashState, &cpHash->b);
    }
    return cpHash;
6.4.4.5  GetCpHash()

This function is used to access a precomputed \( cpHash \).

```c
static TPM2B_DIGEST * GetCpHash(
    COMMAND    *command,
    TPMI_ALG_HASH hashAlg
) {
    TPM2B_DIGEST *cpHash = GetCpHashPointer(command, hashAlg);
    // pAssert(cpHash->t.size != 0);
    return cpHash;
}
```

6.4.4.6  CompareTemplateHash()

This function computes the template hash and compares it to the session \( \text{templateHash} \). It is the hash of the second parameter assuming that the command is TPM2_Create(), TPM2_CreatePrimary(), or TPM2_CreateLoaded().

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>template hash equal to session-&gt;\text{templateHash}</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>template hash not equal to session-&gt;\text{templateHash}</td>
</tr>
</tbody>
</table>

```c
static BOOL CompareTemplateHash(
    COMMAND    *command,  // IN: parsing structure
    SESSION    *session   // IN: session data
) {
    BYTE   *pBuffer = command->parameterBuffer;
    INT32  pSize = command->parameterSize;
    TPM2B_DIGEST tHash;
    UINT16  size;
    // Only try this for the three commands for which it is intended
    if(command->code != TPM_CC_Create && command->code != TPM_CC_CreatePrimary
       #if CC_CreateLoaded
        && command->code != TPM_CC_CreateLoaded
       #endif
    )
    // Assume that the first parameter is a TPM2B and unmarshal the size field
    // Note: this will not affect the parameter buffer and size in the calling
    // function.
    if(UINT16_Unmarshal(&size, &pBuffer, &pSize) != TPM_RC_SUCCESS)
        return FALSE;
    // reduce the space in the buffer.
    // NOTE: this could make pSize go negative if the parameters are not correct but
    // the unmarshaling code does not try to unmarshal if the remaining size is
    // negative.
    pSize -= size;
    // Advance the pointer
    pBuffer += size;
    ```

```
// Get the size of what should be the template
if (UINT16_Unmarshal(&size, &pBuffer, &pSize) != TPM_RC_SUCCESS)
    return FALSE;

// See if this is reasonable
if (size > pSize)
    return FALSE;

// Hash the template data
tHash.t.size = CryptHashBlock(session->authHashAlg, size, pBuffer,
     sizeof(tHash.t.buffer), tHash.t.buffer);
return (MemoryEqual2B(&session->ul.templateHash.b, &tHash.b));

6.4.4.7 CompareNameHash()

This function computes the name hash and compares it to the nameHash in the session data.

BOOL CompareNameHash(
    COMMAND *command, // IN: main parsing structure
    SESSION *session // IN: session structure with nameHash
)
{
    HASH_STATE hashState;
    TPM2B_DIGEST nameHash;
    UINT32 i;
    TPM2B_NAME name;

    // nameHash.t.size = CryptHashStart(&hashState, session->authHashAlg);
    // Add names.
    for (i = 0; i < command->handleNum; i++)
        CryptDigestUpdate2B(&hashState, &EntityGetName(command->handles[i],
            &name)->b);

    // Complete hash.
    CryptHashEnd2B(&hashState, &nameHash.b);

    // and compare
    return MemoryEqual(session->ul.nameHash.t.buffer, nameHash.t.buffer,
        nameHash.t.size);
}

6.4.4.8 CheckPWAuthSession()

This function validates the authorization provided in a PWAP session. It compares the input value to authValue of the authorized entity. Argument sessionIndex is used to get handles handle of the referenced entities from s_inputAuthValues[] and s_associatedHandles[].

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_AUTH_FAIL</td>
<td>authorization fails and increments DA failure count</td>
</tr>
<tr>
<td>TPM_RC_BAD_AUTH</td>
<td>authorization fails but DA does not apply</td>
</tr>
</tbody>
</table>

static TPM_RC
CheckPWAuthSession(
    UINT32 sessionIndex // IN: index of session to be processed
)
{
    TPM2B_AUTH authValue;
    TPM HANDLE associatedHandle = s_associatedHandles[sessionIndex];

    // Strip trailing zeros from the password.
    MemoryRemoveTrailingZeros(&s_inputAuthValues[sessionIndex]);

    // Get the authValue with trailing zeros removed
594       EntityGetAuthValue(associatedHandle, &authValue);
595
596       // Success if the values are identical.
597       if (MemoryEqual2B(&s_inputAuthValues[sessionIndex].b, &authValue.b))
598           { return TPM_RC_SUCCESS; }
599       // if the digests are not identical
600       else {
601           // Invoke DA protection if applicable.
602           return IncrementLockout(sessionIndex);
603       }
604   }

6.4.4.9 ComputeCommandHMAC()

This function computes the HMAC for an authorization session in a command.

607 static TPM2B_DIGEST *
608 ComputeCommandHMAC(
609 COMMAND *command, // IN: primary control structure
610 UINT32 sessionIndex, // IN: index of session to be processed
611 TPM2B_DIGEST *hmac // OUT: authorization HMAC
612 )
613 {
614     TPM2B_TYPE(KEY, (sizeof(AUTH_VALUE) * 2));
615     TPM2B_KEY key;
616     BYTE marshalBuffer[sizeof(TPMA_SESSION)];
617     BYTE *buffer;
618     UINT32 marshalSize;
619     HMAC_STATE hmacState;
620     TPM2B_NONCE *nonceDecrypt;
621     TPM2B_NONCE *nonceEncrypt;
622     SESSION *session;
623     // Determine if extra nonceTPM values are going to be required.
624     // If this is the first session (sessionIndex = 0) and it is
625     // session that uses an HMAC, then check if additional session nonces are to be
626     // included.
627     if (sessionIndex == 0
628         && s_associatedHandles[sessionIndex] != TPM_RH_UNASSIGNED)
629         { if (s_decryptSessionIndex != UNDEFINED_INDEX
630             && s_decryptSessionIndex != sessionIndex)
631             { nonceDecrypt = NULL;
632                 nonceEncrypt = NULL;
633             }
634             // If there is a decrypt session and if this is not the decrypt session,
635             // then an extra nonce may be needed.
636             if (s_decryptSessionIndex != UNDEFINED_INDEX
637                 && s_decryptSessionIndex != sessionIndex)
638                 { nonceDecrypt = &decryptSession->nonceTPM;
639             }
640             // Now repeat for the encrypt session.
641             if (s_encryptSessionIndex != UNDEFINED_INDEX
642                 && s_encryptSessionIndex != sessionIndex
643                 && s_encryptSessionIndex != s_decryptSessionIndex)
644                 { nonceEncrypt = &encryptSession->nonceTPM;
645                     // Have to have the nonce for the encrypt session.
646                     SESSION *encryptSession
647                     = SessionGet(s_sessionHandles[s_encryptSessionIndex]);
648                 }
653   }
654 
655  // Continue with the HMAC processing.
656  session = SessionGet(s_sessionHandles[sessionIndex]);
657 
658  // Generate HMAC key.
659  MemoryCopy2B(&key.b, &session->sessionKey.b, sizeof(key.t.buffer));
660 
661  // Check if the session has an associated handle and if the associated entity
662  // is the one to which the session is bound. If not, add the authValue of
663  // this entity to the HMAC key.
664  // If the session is bound to the object or the session is a policy session
665  // with no authValue required, do not include the authValue in the HMAC key.
666  // Note: For a policy session, its isBound attribute is CLEARED.
667  //
668  // Include the entity authValue if it is needed
669  if(session->attributes.includeAuth == SET)
670  {
671      TPM2B_AUTH authValue;
672      // Get the entity authValue with trailing zeros removed
673      EntityGetAuthValue(s AssociatedHandles[sessionIndex], &authValue);
674      // add the authValue to the HMAC key
675      MemoryConcat2B(&key.b, &authValue.b, sizeof(key.t.buffer));
676  }
677  // if the HMAC key size is 0, a NULL string HMAC is allowed
678  if(key.t.size == 0
679     && s_inputAuthValues[sessionIndex].t.size == 0)
680     {
681        hmac->t.size = 0;
682        return hmac;
683     }
684  // Start HMAC
685  hmac->t.size = CryptHmacStart2B(&hmacState, session->authHashAlg, &key.b);
686  
687      // Add cpHash
688  CryptDigestUpdate2B(&hmacState.hashState,
689      &ComputeCpHash(command, session->authHashAlg) b);
690      // Add nonces as required
691  CryptDigestUpdate2B(&hmacState.hashState, &s_nonceCaller[sessionIndex].b);
692  CryptDigestUpdate2B(&hmacState.hashState, &session->nonceTPM.b);
693  if(nonceDecrypt != NULL)
694     CryptDigestUpdate2B(&hmacState.hashState, &nonceDecrypt b);
695  if(nonceEncrypt != NULL)
696     CryptDigestUpdate2B(&hmacState.hashState, &nonceEncrypt b);
697  
698  // Add sessionAttributes
699  buffer = marshalBuffer;
700  marshalSize = TPMA_SESSION_Marshal(&s_attributes[sessionIndex]),
701        &buffer, NULL);
702  CryptDigestUpdate(&hmacState.hashState, marshalSize, marshalBuffer);
703      // Complete the HMAC computation
704  CryptHmacEnd2B(&hmacState, &hmac->b);
705  
706  return hmac;
707 }

6.4.4.10 CheckSessionHMAC()

This function checks the HMAC of in a session. It uses ComputeCommandHMAC() to compute the
expected HMAC value and then compares the result with the HMAC in the authorization session. The
authorization is successful if they are the same.

If the authorizations are not the same, IncrementLockout() is called. It will return TPM_RC_AUTH_FAIL if
the failure caused the failureCount to increment. Otherwise, it will return TPM_RC_BAD_AUTH.
Error Returns | Meaning
--- | ---
TPM_RC_AUTH_FAIL | authorization failure caused failureCount increment
TPM_RC_BAD_AUTH | authorization failure did not cause failureCount increment

```c
708 static TPM_RC
709 CheckSessionHMAC(
710     COMMAND  *command,  // IN: primary control structure
711     UINT32    sessionIndex  // IN: index of session to be processed
712 )
713 {  // Compute authHMAC
714     TPM2B_DIGEST hmac;  // authHMAC for comparing
715     // Compute authHMAC
716     ComputeCommandHMAC(command, sessionIndex, &hmac);
717     // Compare the input HMAC with the authHMAC computed above.
718     if (!MemoryEqual2B(&s_inputAuthValues[sessionIndex].b, &hmac.b))
719     {
720         // If an HMAC session has a failure, invoke the anti-hammering
721         // if it applies to the authorized entity or the session.
722         // Otherwise, just indicate that the authorization is bad.
723         return IncrementLockout(sessionIndex);
724     }
725     return TPM_RC_SUCCESS;
726 }

6.4.4.11 CheckPolicyAuthSession()

This function is used to validate the authorization in a policy session. This function performs the following comparisons to see if a policy authorization is properly provided. The check are:

a) compare policyDigest in session with authPolicy associated with the entity to be authorized;
b) compare timeout if applicable;
c) compare commandCode if applicable;
d) compare cpHash if applicable; and
e) see if PCR values have changed since computed.

If all the above checks succeed, the handle is authorized. The order of these comparisons is not important because any failure will result in the same error code.

Error Returns | Meaning
--- | ---
TPM_RC_PCR_CHANGED | PCR value is not current
TPM_RC_POLICY_FAIL | policy session fails
TPM_RC_LOCALITY | command locality is not allowed
TPM_RC_POLICY_CC | CC doesn't match
TPM_RC_EXPIRED | policy session has expired
TPM_RC_PP | PP is required but not asserted
TPM_RC_NV_UNAVAILABLE | NV is not available for write
TPM_RC_NV_RATE | NV is rate limiting
```

```c
729 static TPM_RC
730 CheckPolicyAuthSession(
731     COMMAND  *command,  // IN: primary parsing structure
```
UINT32 sessionIndex // IN: index of session to be processed
{
    SESSION *session;
    TPM2B_DIGEST authPolicy;
    TPMI_ALG_HASH policyAlg;
    UINT8 locality;
}

    // Initialize pointer to the authorization session.
    session = SessionGet(s_sessionHandles[sessionIndex]);

    // If the command is TPM2_PolicySecret(), make sure that
    // either password or authValue is required
    if((command->code == TPM_CC_PolicySecret
        && session->attributes.isPasswordNeeded == CLEAR
        && session->attributes.isAuthValueNeeded == CLEAR)
        return TPM_RC_MODE;

    // See if the PCR counter for the session is still valid.
    if(!SessionPCRValueIsCurrent(session))
        return TPM_RC_PCR_CHANGED;

    // Get authPolicy.
    policyAlg = EntityGetAuthPolicy(s_associatedHandles[sessionIndex],
        &authPolicy);

    // Compare authPolicy.
    if(!MemoryEqual2B(&session->u2.policyDigest.b, &authPolicy.b))
        return TPM_RC_POLICY_FAIL;

    // Policy is OK so check if the other factors are correct

    // Compare policy hash algorithm.
    if(policyAlg != session->authHashAlg)
        return TPM_RC_POLICY_FAIL;

    // Compare timeout.
    if(session->timeout != 0)
        {
            // Cannot compare time if clock stop advancing. An TPM_RC_NV_UNAVAILABLE
            // or TPM_RC_NV_RATE error may be returned here. This doesn't mean that
            // a new nonce will be created just that, because TPM time can't advance
            // we can't do time-based operations.
            RETURN_IF_NV_IS_NOT_AVAILABLE;

            if((session->timeout < g_time)
                || (session->epoch != g_timeEpoch))
                return TPM_RC_EXPIRED;
        }

    // If command code is provided it must match
    if(session->commandCode != 0)
        {
            if(session->commandCode != command->code)
                return TPM_RC_POLICY_CC;
        }

    else
        {
            // If command requires a DUP or ADMIN authorization, the session must have
            // command code set.
            AUTH_ROLE role = CommandAuthRole(command->index, sessionIndex);
            if(role == AUTH_ADMIN || role == AUTH_DUP)
                return TPM_RC_POLICY_FAIL;
        }

    // Check command locality.
    {
        BYTE sessionLocality[sizeof(TPMA_LOCALITY)];
        BYTE *buffer = sessionLocality;

        // Get existing locality setting in canonical form
        sessionLocality[0] = 0;
TPMA_LOCALITY_Marshal(&session->commandLocality, &buffer, NULL);

    // See if the locality has been set
    if (sessionLocality[0] != 0)
    {
        // If so, get the current locality
        locality = _plat_LocalityGet();
        if (locality < 5)
        {
            if ((sessionLocality[0] & (1 << locality)) == 0)
            {
                // Could throw an assert here but a locality error is just
                // as good. It just means that, whatever the locality is, it isn't
                // the locality requested so...
                return TPM_RC_LOCALITY;
            }
            else if (locality > 31)
            {
                if (sessionLocality[0] != locality)
                    return TPM_RC_LOCALITY;
            }
            else
            {
                // Could throw an assert here but a locality error is just
                // as good. It just means that, whatever the locality is, it isn't
                // the locality requested so...
                return TPM_RC_LOCALITY;
            }
        }
    }

    // Check physical presence.
    if (session->attributes.isPPRequired == SET
        && !_plat_PhysicalPresenceAsserted())
    {
        return TPM_RC_PP;
    }

    // Compare cpHash/nameHash if defined, or if the command requires an ADMIN or
    // DUP role for this handle.
    if (session->u1.cpHash.b.size != 0)
    {
        BOOL        OK;
        if (session->attributes.isCpHashDefined)
        {
            // Compare cpHash.
            OK = MemoryEqual2B(&session->u1.cpHash.b,
                &ComputeCpHash(command, session->authHashAlg)->b);
        }
        else if (session->attributes.isTemplateSet)
            OK = CompareTemplateHash(command, session);
        else
            OK = CompareNameHash(command, session);
        if (!OK)
            return TPM_RCS_POLICY_FAIL;
    }

    if (session->attributes.checkNvWritten)
    {
        NV_REF           locator;
        NV_INDEX        *nvIndex;
        // If this is not an NV index, the policy makes no sense so fail it.
        if (HandleGetType(s_associatedHandles[sessionIndex]) != TPM_HT_NV_INDEX)
            return TPM_RC_POLICY_FAIL;
        // Get the index data
        nvIndex = NvGetIndexInfo(s_associatedHandles[sessionIndex], &locator);
        // Make sure that the TPMA_WRITTEN_ATTRIBUTE has the desired state
        if (!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))
            return TPM_RC_POLICY_FAIL;
        return TPM_RC_SUCCESS;
    }
}
6.4.4.12 RetrieveSessionData()

This function will unmarshal the sessions in the session area of a command. The values are placed in the arrays that are defined at the beginning of this file. The normal unmarshaling errors are possible.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SUCCESS</td>
<td>unmarshaled without error</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>the number of bytes unmarshaled is not the same as the value for authorizationSize in the command</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
RetrieveSessionData(
    COMMAND *command  // IN: main parsing structure for command
)
{
    int i;
    TPM_RC result;
    SESSION *session;
    TPMA_SESSION sessionAttributes;
    TPM_HT sessionType;
    INT32 sessionIndex;
    TPM_RC errorIndex;

    // s_decryptSessionIndex = UNDEFINED_INDEX;
    s_encryptSessionIndex = UNDEFINED_INDEX;
    s_auditSessionIndex = UNDEFINED_INDEX;

    for(sessionIndex = 0; command->authSize > 0; sessionIndex++)
    {
        errorIndex = TPM_RC_S + g_rcIndex[sessionIndex];

        // If maximum allowed number of sessions has been parsed, return a size
        // error with a session number that is larger than the number of allowed
        // sessions
        if(sessionIndex == MAX_SESSION_NUM)
            return TPM_RC_SIZE + errorIndex;

        // make sure that the associated handle for each session starts out
        // unassigned
        s_associatedHandles[sessionIndex] = TPM_RH_UNASSIGNED;

        // First parameter: Session handle.
        result = TPMI_SH_AUTH_SESSION_Unmarshal(
            &s_sessionHandles[sessionIndex],
            &command->parameterBuffer,
            &command->authSize, TRUE);
        if(result != TPM_RC_SUCCESS)
            return result + TPM_RC_S + g_rcIndex[sessionIndex];

        // Second parameter: Nonce.
        result = TPM2B_NONCE_Unmarshal(&s_nonceCaller[sessionIndex],
            &command->parameterBuffer,
            &command->authSize);
        if(result != TPM_RC_SUCCESS)
            return result + TPM_RC_S + g_rcIndex[sessionIndex];

        // Third parameter: sessionAttributes.
        result = TPMA_SESSION_Unmarshal(&s_attributes[sessionIndex],
            &command->parameterBuffer,
            &command->authSize);
        if(result != TPM_RC_SUCCESS)
            return result + TPM_RC_S + g_rcIndex[sessionIndex];

        // Fourth parameter: authValue (PW or HMAC).
        result = TPM2B_AUTH_Unmarshal(&s_inputAuthValues[sessionIndex],
            &command->parameterBuffer,
            &command->authSize);
```

Family “2.0” TCG Public Review Page 211
Level 00 Revision 01.50 Copyright © TCG 2006-2019 September 18, 2018
if(result != TPM_RC_SUCCESS)
    return result + errorIndex;

sessionAttributes = s_attributes[sessionIndex];
if(s_sessionHandles[sessionIndex] == TPM_RS_PW)
{
    // A FWAP session needs additional processing.
    // Can't have any attributes set other than continueSession bit
    if(IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, encrypt)
        || IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, decrypt)
        || IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, audit)
        || IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, auditExclusive)
        || IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, auditReset))
        return TPM_RCS_ATTRIBUTES + errorIndex;
    // The nonce size must be zero.
    if(s_nonceCaller[sessionIndex].t.size != 0)
        return TPM_RCS_NONCE + errorIndex;
    continue;
}
// For not password sessions...
// Find out if the session is loaded.
if(!SessionIsLoaded(s_sessionHandles[sessionIndex]))
    return TPM_RC_REFERENCE_S0 + sessionIndex;
sessionType = HandleGetType(s_sessionHandles[sessionIndex]);
session = SessionGet(s_sessionHandles[sessionIndex]);

// Check if the session is an HMAC/policy session.
if((session->attributes.isPolicy == SET
    && sessionType == TPM_HT_HMAC_SESSION)
    || (session->attributes.isPolicy == CLEAR
        && sessionType == TPM_HT_POLICY_SESSION))
    return TPM_RCS_HANDLE + errorIndex;
// Check that this handle has not previously been used.
for(i = 0; i < sessionIndex; i++)
{
    if(s_sessionHandles[i] == s_sessionHandles[sessionIndex])
        return TPM_RCS_HANDLE + errorIndex;
}
// If the session is used for parameter encryption or audit as well, set
// the corresponding indexes.
// First process decrypt.
if(IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, decrypt))
{
    // Check if the commandCode allows command parameter encryption.
    if(EncryptSize(command->index) == 0)
        return TPM_RCS_ATTRIBUTES + errorIndex;
    // Decrypt attribute can only appear in one session
    if(s_decryptSessionIndex != UNDEFINED_INDEX)
        return TPM_RCS_ATTRIBUTES + errorIndex;
    // Can't decrypt if the session's symmetric algorithm is TPM_ALG_NULL
    if(session->symmetric.algorithm == TPM_ALG_NULL)
        return TPM_RCS_SYMMETRIC + errorIndex;
    // All checks passed, so set the index for the session used to decrypt
    // a command parameter.
    s_decryptSessionIndex = sessionIndex;
}
// Now process encrypt.
if(IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, encrypt))
{
    // Check if the commandCode allows response parameter encryption.
    if(EncryptSize(command->index) == 0)
        return TPM_RCS_ATTRIBUTES + errorIndex;
    // Encrypt attribute can only appear in one session
    if(s_encryptSessionIndex != UNDEFINED_INDEX)
        return TPM_RCS_ATTRIBUTES + errorIndex;
// Can't encrypt if the session's symmetric algorithm is TPM_ALG_NULL
if (session->symmetric.algorithm == TPM_ALG_NULL)
    return TPM_RC_SYMMETRIC + errorIndex;
// All checks passed, so set the index for the session used to encrypt
// a response parameter.
_s_encryptSessionIndex = sessionIndex;

// At last process audit.
if (IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, audit))
{
    // Audit attribute can only appear in one session.
    if (_s_auditSessionIndex != UNDEFINED_INDEX)
        return TPM_RC_ATTRIBUTES + errorIndex;
    // An audit session can not be policy session.
    if (HandleGetType(s_sessionHandles[sessionIndex])
        == TPM_HT_POLICY_SESSION)
        return TPM_RC_ATTRIBUTES + errorIndex;
    // If this is a reset of the audit session, or the first use
    // of the session as an audit session, it doesn't matter what
    // the exclusive state is. The session will become exclusive.
    if (!IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, auditReset)
        && session->attributes.isAudit == SET)
    {
        // Not first use or reset. If auditExclusive is SET, then this
        // session must be the current exclusive session.
        if (IS_ATTRIBUTE(sessionAttributes, TPMA_SESSION, auditExclusive)
            && g_exclusiveAuditSession != s_sessionHandles[sessionIndex])
            return TPM_RC_EXCLUSIVE;
    }
    _s_auditSessionIndex = sessionIndex;
}

// Initialize associated handle as undefined. This will be changed when
// the handles are processed.
_s_associatedHandles[sessionIndex] = TPM_RH_UNASSIGNED;

// command->sessionNum = sessionIndex;
return TPM_RC_SUCCESS;

6.4.4.13 CheckLockedOut()

This function checks to see if the TPM is in lockout. This function should only be called if the entity being
checked is subject to DA protection. The TPM is in lockout if the NV is not available and a DA write is
pending. Otherwise the TPM is locked out if checking for lockoutAuth (lockoutAuthCheck == TRUE) and
use of lockoutAuth is disabled, or failedTries >= maxTries

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>NV is rate limiting</td>
</tr>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>NV is not available at this time</td>
</tr>
<tr>
<td>TPM_RC_LOCKOUT</td>
<td>TPM is in lockout</td>
</tr>
</tbody>
</table>

static TPM_RC
CheckLockedOut(
    BOOL             lockoutAuthCheck // IN: TRUE if checking is for lockoutAuth
)
{
    // If NV is unavailable, and current cycle state recorded in NV is not
    // SU_NONE_VALUE, refuse to check any authorization because we would
    // not be able to handle a DA failure.
    if (!NV_ISAVAILABLE && NV_IS_ORDERLY)
        return g_NvStatus;

    // If NV is unavailable, and current cycle state recorded in NV is not
    // SU_NONE_VALUE, refuse to check any authorization because we would
    // not be able to handle a DA failure.
    if (!NV_IS_AVAILABLE && NV_IS_ORDERLY)
// Check if DA info needs to be updated in NV.
if (s_DAPendingOnNV)
{
    // If NV is accessible,
    RETURN_IF_NV_IS_NOT_AVAILABLE;
    // ... write the pending DA data and proceed.
    NV_SYNC_PERSISTENT(lockOutAuthEnabled);
    NV_SYNC_PERSISTENT(failedTries);
    s_DAPendingOnNV = FALSE;
}

// Lockout is in effect if checking for lockoutAuth and use of lockoutAuth
// is disabled...
if (lockoutAuthCheck)
{
    if (gp.lockOutAuthEnabled == FALSE)
        return TPM_RC_LOCKOUT;
    else
    {
        // ... or if the number of failed tries has been maxed out.
        if (gp.failedTries >= gp.maxTries)
            return TPM_RC_LOCKOUT;

        #if USE_DA_USED
        // If the daUsed flag is not SET, then no DA validation until the
        // daUsed state is written to NV
        if (!g_daUsed)
        {
            RETURN_IF_NV_IS_NOT_AVAILABLE;
            g_daUsed = TRUE;
            gp.orderlyState = SU_DA_USED_VALUE;
            NV_SYNC_PERSISTENT(orderlyState);
            return TPM_RC_RETRY;
        }
        #endif
    }
}

return TPM_RC_SUCCESS;

6.4.4.14 CheckAuthSession()

This function checks that the authorization session properly authorizes the use of the associated handle.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_LOCKOUT</td>
<td>entity is protected by DA and TPM is in lockout, or TPM is locked out on NV update pending on DA parameters</td>
</tr>
<tr>
<td>TPM_RC_PP</td>
<td>Physical Presence is required but not provided</td>
</tr>
<tr>
<td>TPM_RC_AUTH_FAIL</td>
<td>HMAC or PW authorization failed with DA side-effects (can be a policy session)</td>
</tr>
<tr>
<td>TPM_RC_BAD_AUTH</td>
<td>HMAC or PW authorization failed without DA side-effects (can be a policy session)</td>
</tr>
<tr>
<td>TPM_RC_POLICY_FAIL</td>
<td>if policy session fails</td>
</tr>
<tr>
<td>TPM_RC_POLICY_CC</td>
<td>command code of policy was wrong</td>
</tr>
<tr>
<td>TPM_RC_EXPIRED</td>
<td>the policy session has expired</td>
</tr>
<tr>
<td>TPM_RC_PCR</td>
<td>???</td>
</tr>
<tr>
<td>TPM_RC_AUTH_UNAVAILABLE</td>
<td>authValue or authPolicy unavailable</td>
</tr>
</tbody>
</table>

static TPM_RC
CheckAuthSession(  
COMMAND *command,   // IN: primary parsing structure  
UINT32 sessionIndex // IN: index of session to be processed  
)

{  
TPM_RC result = TPM_RC_SUCCESS;
SESSION *session = NULL;
TPM_HANDLE sessionHandle = s_sessionHandles[sessionIndex];
TPM_HANDLE associatedHandle = s_associatedHandles[sessionIndex];
TPM_HT sessionHandleType = HandleGetType(sessionHandle);

// pAssert(sessionHandle != TPM_RH_UNASSIGNED);

// Take care of physical presence
if(associatedHandle == TPM_RH_PLATFORM)
{
    // If the physical presence is required for this command, check for PP
    // assertion. If it isn’t asserted, no point going any further.
    if(PhysicalPresenceIsRequired(command->index)  
        && !_plat__PhysicalPresenceAsserted())
        return TPM_RC_PP;
}
if(sessionHandle != TPM_RS_PW)
{
    session = SessionGet(sessionHandle);
    // Set includeAuth to indicate if DA checking will be required and if the
    // authValue will be included in any HMAC.
    if(sessionHandleType == TPM_HT_POLICY_SESSION)
    {
        // For a policy session, will check the DA status of the entity if either
        // isAuthValueNeeded or isPasswordNeeded is SET
        session->attributes.includeAuth =  
            session->attributes.isAuthValueNeeded  
            || session->attributes.isPasswordNeeded;
    }
    else
    {
        // For an HMAC session, need to check unless the session
        // is bound.
        session->attributes.includeAuth =  
            !IsSessionBindEntity(s_associatedHandles[sessionIndex], session);
    }

    // If the authorization session is going to use an authValue, then make sure
    // that access to that authValue isn't locked out.
    // Note: session == NULL for a PW session.
    if(session == NULL || session->attributes.includeAuth)
    {
        // See if entity is subject to lockout.
        if(!IsDAExempted(associatedHandle))
        {
            // See if in lockout
            result = CheckLockedOut(associatedHandle == TPM_RH_LOCKOUT);
            if(result != TPM_RC_SUCCESS)
                return result;
        }
    }
}

// Policy or HMAC+PW?
if(sessionHandleType != TPM_HT_POLICY_SESSION)
{
    // for non-policy session make sure that a policy session is not required
    if(IsPolicySessionRequired(command->index, sessionIndex))
        return TPM_RC_AUTH_TYPE;

    // The authValue must be available.
    // Note: The authValue is going to be "used" even if it is an EmptyAuth.
// and the session is bound.
if (!IsAuthValueAvailable(associatedHandle, command->index, sessionIndex))
    return TPM_RC_AUTH_UNAVAILABLE;
else {
    // ... see if the entity has a policy, ...
    // Note: IsAuthPolicyAvailable will return FALSE if the sensitive area of the
    // object is not loaded
    if (!IsAuthPolicyAvailable(associatedHandle, command->index, sessionIndex))
        return TPM_RC_AUTH_UNAVAILABLE;
    // ... and check the policy session.
    result = CheckPolicyAuthSession(command, sessionIndex);
    if (result != TPM_RC_SUCCESS)
        return result;
}

// Check authorization according to the type
if (session == NULL || session->attributes.isPasswordNeeded == SET)
    result = CheckPWAuthSession(sessionIndex);
else
    result = CheckSessionHMAC(command, sessionIndex);

// Do processing for PIN Indexes are only three possibilities for 'result' at
// this point: TPM_RC_SUCCESS, TPM_RC_AUTH_FAIL, and TPM_RC_BAD_AUTH.
// For all these cases, we have to process a PIN index if the
// authValue of the index was used for authorization.
// See if we need to do anything to a PIN index
if (TPM_HT_NV_INDEX == HandleGetType(associatedHandle)) {
    NV_REF locator;
    NV_INDEX *nvIndex = NvGetIndexInfo(associatedHandle, &locator);
    NV_PIN pinData;
    TPMA_NV nvAttributes;

    // pAssert(nvIndex != NULL);
    nvAttributes = nvIndex->publicArea.attributes;
    // If this is a PIN FAIL index and the value has been written
    // then we can update the counter (increment or clear)
    if (IsNvPinFailIndex(nvAttributes) && IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITTEN))
        { pinData.intVal = NvGetUINT64Data(nvIndex, locator);
          if (result != TPM_RC_SUCCESS)
              pinData.pin.pinCount++;
          else
              pinData.pin.pinCount = 0;
          NvWriteUINT64Data(nvIndex, pinData.intVal);
        }
    else
        { pinData.intVal = NvGetUINT64Data(nvIndex, locator);
          pinData.pin.pinCount++;
          NvWriteUINT64Data(nvIndex, pinData.intVal);
        }
}

#endif
6.4.4.15 CheckCommandAudit()

This function is called before the command is processed if audit is enabled for the command. It will check to see if the audit can be performed and will ensure that the cpHash is available for the audit.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>NV is not available for write</td>
</tr>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>NV is rate limiting</td>
</tr>
</tbody>
</table>

```c
static TPM_RC CheckCommandAudit(
    COMMAND *command)
{
    // If the audit digest is clear and command audit is required, NV must be
    // available so that TPM2_GetCommandAuditDigest() is able to increment
    // audit counter. If NV is not available, the function bails out to prevent
    // the TPM from attempting an operation that would fail anyway.
    if (gr.commandAuditDigest.t.size == 0
        || GetCommandCode(command->index) == TPM_CC_GetCommandAuditDigest)
    {
        return TPM_RC_SUCCESS;
    }
    ComputeCpHash(command, gp.auditHashAlg);
}
```

6.4.4.16 ParseSessionBuffer()

This function is the entry function for command session processing. It iterates sessions in session area and reports if the required authorization has been properly provided. It also processes audit session and passes the information of encryption sessions to parameter encryption module.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>various</td>
<td>parsing failure or authorization failure</td>
</tr>
</tbody>
</table>

```c
TPM_RC ParseSessionBuffer(
    COMMAND *command)
{
    TPM_RC result;
    UINT32 i;
    INT32 size = 0;
    TPM2B_AUTH extraKey;
    UINT32 sessionIndex;
    TPM_RC errorIndex;
    SESSION *session = NULL;
    // Check if a command allows any session in its session area.
    if (!IsSessionAllowed(command->index))
    {
        return TPM_RC_AUTH_CONTEXT;
    }
    // Default-initialization.
    command->sessionNum = 0;
    result = RetrieveSessionData(command);
    if (result != TPM_RC_SUCCESS)
    {
        return result;
    }
}
```
// There is no command in the TPM spec that has more handles than MAX_SESSION_NUM.
*pAssert(command->handleNum <= MAX_SESSION_NUM);

// Associate the session with an authorization handle.
for (i = 0; i < command->handleNum; i++)
{
    if (CommandAuthRole(command->index, i) != AUTH_NONE)
    {
        // If the received session number is less than the number of handles
        // that requires authorization, an error should be returned.
        // Note: for all the TPM 2.0 commands, handles requiring
        // authorization come first in a command input and there are only ever
        // two values requiring authorization
        if (i > (command->sessionNum - 1))
            return TPM_RC_AUTH_MISSING;
        // Record the handle associated with the authorization session
        s_associatedHandles[i] = command->handles[i];
    }
}

// Consistency checks are done first to avoid authorization failure when the
// command will not be executed anyway.
for (sessionIndex = 0; sessionIndex < command->sessionNum; sessionIndex++)
{
    errorIndex = TPM_RC_S + g_rcIndex[sessionIndex];
    // PW session must be an authorization session
    if (s_sessionHandles[sessionIndex] == TPM_RS_PW)
    {
        if (s_associatedHandles[sessionIndex] == TPM_RH_UNASSIGNED)
            return TPM_RCS_HANDLE + errorIndex;
        // a password session can't be audit, encrypt or decrypt
        if (IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, audit)
            || IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, encrypt)
            || IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, decrypt))
            return TPM_RCS_ATTRIBUTES + errorIndex;
        session = NULL;
    }
    else
    {
        session = SessionGet(s_sessionHandles[sessionIndex]);
        // A trial session can not appear in session area, because it cannot
        // be used for authorization, audit or encrypt/decrypt.
        if (session->attributes.isTrialPolicy == SET)
            return TPM_RCS_ATTRIBUTES + errorIndex;
        // See if the session is bound to a DA protected entity
        // NOTE: Since a policy session is never bound, a policy is still
        // usable even if the object is DA protected and the TPM is in
        // lockout.
        if (session->attributes.isDaBound == SET)
        {
            result = CheckLockedOut(session->attributes.isLockoutBound == SET);
            if (result != TPM_RC_SUCCESS)
                return result;
        }
        // If this session is for auditing, make sure the cpHash is computed.
        if (IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, audit))
            ComputeCpHash(command, session->authHashAlg);
    }
    // if the session has an associated handle, check the authorization
    if (s_associatedHandles[sessionIndex] != TPM_RH_UNASSIGNED)
    {
        result = CheckAuthSession(command, sessionIndex);
        if (result != TPM_RC_SUCCESS)
return RcSafeAddToResult(result, errorIndex);
}
else {
    // a session that is not for authorization must either be encrypt,
    // decrypt, or audit
    if(!IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, audit)  
        && !IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, encrypt)  
        && !IS_ATTRIBUTE(s_attributes[sessionIndex], TPMA_SESSION, decrypt))
        return TPM_RCS_ATTRIBUTES + errorIndex;

    // no authValue included in any of the HMAC computations
    pAssert(session != NULL);
    session->attributes.includeAuth = CLEAR;

    // check HMAC for encrypt/decrypt/audit only sessions
    result = CheckSessionHMAC(command, sessionIndex);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, errorIndex);
}
#endif

// Decrypt the first parameter if applicable. This should be the last operation
// in session processing.
// If the encrypt session is associated with a handle and the handle's
// authValue is available, then authValue is concatenated with sessionKey to
// generate encryption key, no matter if the handle is the session bound entity
// or not.
if(s_decryptSessionIndex != UNDEFINED_INDEX)
{
    // If this is an authorization session, include the authValue in the
    // generation of the decryption key
    if(s_associatedHandles[s_decryptSessionIndex] != TPM_RH_UNASSIGNED)
    {
        EntityGetAuthValue(s_associatedHandles[s_decryptSessionIndex],
            &extraKey);
    }
    else
    {
        extraKey.b.size = 0;
    }

    size = DecryptSize(command->index);
    result = CryptParameterDecryption(s_sessionHandles[s_decryptSessionIndex],
        &s_nonceCaller[s_decryptSessionIndex].b,
        command->parameterSize, (UINT16)size,
        &extraKey,
        command->parameterBuffer);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result,  
            TPM_RC_S + g_rcIndex[s_decryptSessionIndex]);
}
return TPM_RC_SUCCESS;
6.4.4.17 CheckAuthNoSession()

Function to process a command with no session associated. The function makes sure all the handles in the command require no authorization.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_AUTH_MISSING</td>
<td>failure - one or more handles require authorization</td>
</tr>
</tbody>
</table>

static TPM_RC
CheckAuthNoSession(
    COMMAND         *command      // IN: command parsing structure
)
{
    UINT32 i;
    TPM_RC result = TPM_RC_SUCCESS;
    // Check if the command requires authorization
    for(i = 0; i < command->handleNum; i++)
    {
        if(CommandAuthRole(command->index, i) != AUTH_NONE)
            return TPM_RC_AUTH_MISSING;
    }
    #ifdef TPM_CC_GetCommandAuditDigest
    // Check if the command should be audited.
    if(CommandAuditIsRequired(command->index))
    {
        result = CheckCommandAudit(command);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    #endif
    // Initialize number of sessions to be 0
    command->sessionNum = 0;
    return TPM_RC_SUCCESS;
}

6.4.5 Response Session Processing

6.4.5.1 Introduction

The following functions build the session area in a response, and handle the audit sessions (if present).

6.4.5.2 ComputeRpHash()

Function to compute rpHash (Response Parameter Hash). The rpHash is only computed if there is an HMAC authorization session and the return code is TPM_RC_SUCCESS.

static TPM2B_DIGEST *
ComputeRpHash(
    COMMAND         *command,      // IN: command structure
    TPM_ALG_ID hashAlg          // IN: hash algorithm to compute rpHash
)
{
    TPM2B_DIGEST *rpHash = GetRpHashPointer(command, hashAlg);
    HASH_STATE hashState;
    // rpHash := hash(responseCode || commandCode || parameters)
// Initiate hash creation.
rpHash->t.size = CryptHashStart(&hashState, hashAlg);

// Add hash constituents.
CryptDigestUpdateInt(&hashState, sizeof(TPM_RC), TPM_RC_SUCCESS);
CryptDigestUpdateInt(&hashState, sizeof(TPM_CC), command->code);
CryptDigestUpdate(&hashState, command->parameterSize,
    command->parameterBuffer);

// Complete hash computation.
CryptHashEnd2B(&hashState, &rpHash->b);
}
return rpHash;

6.4.5.3 InitAuditSession()

This function initializes the audit data in an audit session.

static void
InitAuditSession(
    SESSION *session // session to be initialized
)
{
    // Mark session as an audit session.
    session->attributes.isAudit = SET;
    // Audit session can not be bound.
    session->attributes.isBound = CLEAR;
    // Size of the audit log is the size of session hash algorithm digest.
    session->u2.auditDigest.t.size = CryptHashGetDigestSize(session->authHashAlg);
    // Set the original digest value to be 0.
    MemorySet(&session->u2.auditDigest.t.buffer,
        0,
        session->u2.auditDigest.t.size);
    return;
}

6.4.5.4 UpdateAuditDigest

Function to update an audit digest.

static void
UpdateAuditDigest(
    COMMAND *command,
    TPMI_ALG_HASH hashAlg,
    TPM2B_DIGEST *digest
)
{
    // Complete hash computation.
    CryptDigestUpdate2B(&hashState, &digest->b);
}

// digestNew := hash (digestOld || cpHash || rpHash)
// Start hash computation.
digest->t.size = CryptHashStart(&hashState, hashAlg);
// Add old digest.
CryptDigestUpdate2B(&hashState, &digest->b);
// Add cpHash
6.4.5.5 Audit()

This function updates the audit digest in an audit session.

```c
static void
Audit(
    COMMAND *command, // IN: primary control structure
    SESSION *auditSession // IN: loaded audit session
)
{
    UpdateAuditDigest(command, auditSession->authHashAlg,
        &auditSession->u2.auditDigest);
    return;
}
```

6.4.5.6 CommandAudit()

This function updates the command audit digest.

```c
static void
CommandAudit(
    COMMAND *command // IN:
)
{
    // If the digest.size is one, it indicates the special case of changing
    // the audit hash algorithm. For this case, no audit is done on exit.
    // NOTE: When the hash algorithm is changed, g_updateNV is set in order to
    // force an update to the NV on exit so that the change in digest will
    // be recorded. So, it is safe to exit here without setting any flags
    // because the digest change will be written to NV when this code exits.
    if(gr.commandAuditDigest.t.size == 1)
    {
        gr.commandAuditDigest.t.size = 0;
        return;
    }
    // If the digest size is zero, need to start a new digest and increment
    // the audit counter.
    if(gr.commandAuditDigest.t.size == 0)
    {
        gr.commandAuditDigest.t.size = CryptHashGetDigestSize(gp.auditHashAlg);
        MemorySet(gr.commandAuditDigest.t.buffer,
            0,
            gr.commandAuditDigest.t.size);
        // Bump the counter and save its value to NV.
        gp.auditCounter++;
        NV_SYNC_PERSISTENT(auditCounter);
    }
    UpdateAuditDigest(command, gp.auditHashAlg, &gr.commandAuditDigest);
    return;
}
```
6.4.5.7 UpdateAuditSessionStatus()

Function to update the internal audit related states of a session. It
a) initializes the session as audit session and sets it to be exclusive if this is the first time it is used for
audit or audit reset was requested;
b) reports exclusive audit session;
c) extends audit log; and

d) clears exclusive audit session if no audit session found in the command.

1513 static void
1514 UpdateAuditSessionStatus(
1515 COMMAND *command   // IN: primary control structure
1516 )
1517 {
1518   UINT32 i;
1519   TPM_HANDLE auditSession = TPM_RH_UNASSIGNED;
1520   // Iterate through sessions
1521   for(i = 0; i < command->sessionNum; i++)
1522   {
1523     SESSION *session;
1524     // FW session do not have a loaded session and can not be an audit
1525     // session either. Skip it.
1526     if(s_sessionHandles[i] == TPM_RS_PW)
1527       continue;
1528     session = SessionGet(s_sessionHandles[i]);
1529     // If a session is used for audit
1530     if(IS_ATTRIBUTE(s_attributes[i], TPMA_SESSION, audit))
1531       {
1532         // An audit session has been found
1533         auditSession = s_sessionHandles[i];
1534         // If the session has not been an audit session yet, or
1535         // the auditSetting bits indicate a reset, initialize it and set
1536         // it to be the exclusive session
1537         if(session->attributes.isAudit == CLEAR
1538           || IS_ATTRIBUTE(s_attributes[i], TPMA_SESSION, auditReset))
1539           {
1540             InitAuditSession(session);
1541             g_exclusiveAuditSession = auditSession;
1542           }
1543       }
1544     else
1545       {
1546         
1547         // Check if the audit session is the current exclusive audit
1548         // session and, if not, clear previous exclusive audit session.
1549         if(g_exclusiveAuditSession != auditSession)
1550           g_exclusiveAuditSession = TPM_RH_UNASSIGNED;
1551       }
1552       // Report audit session exclusivity.
1553       if(g_exclusiveAuditSession == auditSession)
1554         {
1555           SET_ATTRIBUTE(s_attributes[i], TPMA_SESSION, auditExclusive);
1556         }
1557     else
1558       {
1559         CLEAR_ATTRIBUTE(s_attributes[i], TPMA_SESSION, auditExclusive);
1560       }
1561     // Extend audit log.
1562     Audit(command, session);
1563   }
1564 }
1566 } // If no audit session is found in the command, and the command allows
1567 // a session then, clear the current exclusive
1568 // audit session.
1569 if (auditSession == TPM_RH_UNASSIGNED && IsSessionAllowed(command->index))
1570 {
1571     g_exclusiveAuditSession = TPM_RH_UNASSIGNED;
1572 }
1573 return;
1574 }
1575 }

6.4.5.8 ComputeResponseHMAC()

Function to compute HMAC for authorization session in a response.

1576 static void
1577 ComputeResponseHMAC(
1578     COMMAND *command, // IN: command structure
1579     UINT32 sessionIndex, // IN: session index to be processed
1580     SESSION *session, // IN: loaded session
1581     TPM2B_DIGEST *hmac // OUT: authHMAC
1582 )
1583 {
1584     TPM2B_TYPE(KEY, (sizeof(AUTH_VALUE) * 2));
1585     TPM2B_KEY key; // HMAC key
1586     BYTE marshalBuffer[ sizeof(TPMA_SESSION)];
1587     BYTE *buffer;
1588     UINT32 marshalSize;
1589     HMAC_STATE hmacState;
1590     TPM2B_DIGEST *rpHash = ComputeRpHash(command, session->authHashAlg);
1591     // Generate HMAC key
1592     MemoryCopy2B(&key.b, &session->sessionKey.b, sizeof(key.t.buffer));
1593     // Add the object authValue if required
1594     if (session->attributes.includeAuth == SET)
1595         {
1596             // Note: includeAuth may be SET for a policy that is used in
1597             // UndefineSpaceSpecial(). At this point, the Index has been deleted
1598             // so the includeAuth will have no meaning. However, the
1599             // s_associatedHandles[] value for the session is now set to TPM_RH_NULL so
1600             // this will return the authValue associated with TPM_RH_NULL and that is
1601             // and empty buffer.
1602             TPM2B_AUTH authValue;
1603             // Get the authValue with trailing zeros removed
1604             EntityGetAuthValue(s_associatedHandles[sessionIndex], &authValue);
1605             // Add it to the key
1606             MemoryConcat2B(&key.b, &authValue.b, sizeof(key.t.buffer));
1607         }
1608     // if the HMAC key size is 0, the response HMAC is computed according to the
1609     // input HMAC
1610     if (key.t.size == 0)
1611         {
1612             // Start HMAC computation.
1613             hmac->t.size = CryptoHmacStart2B(&hmacState, session->authHashAlg, &key.b);
1614             // Add hash components.

CryptDigestUpdate2B(&hmacState.hashState, &rpHash->b);
CryptDigestUpdate2B(&hmacState.hashState, &session->nonceTPM.b);
CryptDigestUpdate2B(&hmacState.hashState, &s_nonceCaller[sessionIndex].b);

// Add session attributes.
buffer = marshalBuffer;
marshalSize = TPMA_SESSION_Marshal(&s_attributes[sessionIndex], &buffer, NULL);
CryptDigestUpdate(&hmacState.hashState, marshalSize, marshalBuffer);

// Finalize HMAC.
CryptHmacEnd2B(&hmacState, &hmac->b);
return;

6.4.5.9 UpdateInternalSession()

Updates internal sessions:

a) Restarts session time.

b) Clears a policy session since nonce is rolling.

static void UpdateInternalSession(
  SESSION *session,
  UINT32 i
)[/]

  // If nonce is rolling in a policy session, the policy related data
  // will be re-initialized.
  if(HandleGetType(s_sessionHandles[i]) == TPM_HT_POLICY_SESSION
    && IS_ATTRIBUTE(s_attributes[i], TPMA_SESSION, continueSession))
    {
      // When the nonce rolls it starts a new timing interval for the
      // policy session.
      SessionResetPolicyData(session);
      SessionSetStartTime(session);
    }
    return;

6.4.5.10 BuildSingleResponseAuth()

Function to compute response HMAC value for a policy or HMAC session.

static TPM2B_NONCE *BuildSingleResponseAuth(
  COMMAND *command,
  UINT32 sessionIndex,
  TPM2B_AUTH *auth)[/]

  // Fill in policy/HMAC based session response.
  SESSION *session = SessionGet(s_sessionHandles[sessionIndex]);
  // If the session is a policy session with isPasswordNeeded SET, the
  // authorization field is empty.
  if(HandleGetType(s_sessionHandles[sessionIndex]) == TPM_HT_POLICY_SESSION
    && session->attributes.isPasswordNeeded == SET)
    auth->t.size = 0;
  else
    // Compute response HMAC.

ComputeResponseHMAC(command, sessionIndex, session, auth);
UpdateInternalSession(session, sessionIndex);
return &session->nonceTPM;

6.4.5.11 UpdateAllNonceTPM()

Updates TPM nonce for all sessions in command.

static void UpdateAllNonceTPM(COMMAND *command)
{
    UINT32 i;
    SESSION *session;
    for(i = 0; i < command->sessionNum; i++)
    {
        // If not a PW session, compute the new nonceTPM.
        if(s_sessionHandles[i] != TPM_RS_PW)
        {
            session = SessionGet(s_sessionHandles[i]);
            // Update nonceTPM in both internal session and response.
            CryptRandomGenerate(session->nonceTPM.t.size,
                               session->nonceTPM.t.buffer);
        }
    }
    return;
}

6.4.5.12 BuildResponseSession()

Function to build Session buffer in a response. The authorization data is added to the end of command->responseBuffer. The size of the authorization area is accumulated in command->authSize. When this is called, command->responseBuffer is pointing at the next location in the response buffer to be filled. This is where the authorization sessions will go, if any. command->parameterSize is the number of bytes that have been marshaled as parameters in the output buffer.

void BuildResponseSession(COMMAND *command)
{
pAssert(command->authSize == 0);
    // Reset the parameter buffer to point to the start of the parameters so that
    // there is a starting point for any rpHash that might be generated and so there
    // is a place where parameter encryption would start
    command->parameterBuffer = command->responseBuffer - command->parameterSize;
    // Session nonces should be updated before parameter encryption
    if(command->tag == TPM_ST_SESSIONS)
    {
        UpdateAllNonceTPM(command);
        // Encrypt first parameter if applicable. Parameter encryption should
        // happen after nonce update and before any rpHash is computed.
        // If the encrypt session is associated with a handle, the authValue of
        // this handle will be concatenated with sessionKey to generate
// encryption key, no matter if the handle is the session bound entity
// or not. The authValue is added to sessionKey only when the authValue
// is available.
if(s_encryptSessionIndex != UNDEFINED_INDEX)
{
    UINT32          size;
    TPM2B_AUTH      extraKey;
    //
    extraKey.b.size = 0;
    // If this is an authorization session, include the authValue in the
    // generation of the encryption key
    if(s_associatedHandles[s_encryptSessionIndex] != TPM_RH_UNASSIGNED)
    {
        EntityGetAuthValue(s_associatedHandles[s_encryptSessionIndex],
                           &extraKey);
    }
    size = EncryptSize(command->index);
    CryptParameterEncryption(s_sessionHandles[s_encryptSessionIndex],
                             &s_nonceCaller[s_encryptSessionIndex].b,
                             (UINT16)size,
                             &extraKey,
                             command->parameterBuffer);
}
// Audit sessions should be processed regardless of the tag because
// a command with no session may cause a change of the exclusivity state.
UpdateAuditSessionStatus(command);
#if CC_GetCommandAuditDigest
// Command Audit
if(CommandAuditIsRequired(command->index))
    CommandAudit(command);
#endif
// Process command with sessions.
if(command->tag == TPM_ST_SESSIONS)
{
    UINT32           i;
    //
    pAssert(command->sessionNum > 0);
    // Iterate over each session in the command session area, and create
    // corresponding sessions for response.
    for(i = 0; i < command->sessionNum; i++)
    {
        TPM2B_NONCE     *nonceTPM;
        TPM2B_DIGEST    responseAuth;
        // Make sure that continueSession is SET on any Password session.
        // This makes it marginally easier for the management software
        // to keep track of the closed sessions.
        if(s_sessionHandles[i] == TPM_RS_PW)
        {
            SET_ATTRIBUTE(s_attributes[i], TPMA_SESSION, continueSession);
            responseAuth.t.size = 0;
            nonceTPM = (TPM2B_NONCE *)&responseAuth;
        }
        else
        {
            // Compute the response HMAC and get a pointer to the nonce used.
            // This function will also update the values if needed. Note, the
            nonceTPM = BuildSingleResponseAuth(command, i, &responseAuth);
        }
        command->authSize += TPM2B_NONCE.Marshal(nonceTPM,
                                                  &command->responseBuffer,
                                                  NULL);
        command->authSize += TPMA_SESSION.Marshal(&s_attributes[i],
                                                  &command->responseBuffer,
                                                  NULL);
    }
trustedPlatformModuleLib

Part 4: Supporting Routines

Page 228

TCG Public Review

Family “2.0”

September 18, 2018

Copyright © TCG 2006-2019

Level 00 Revision 01.50

command->authSize += TPM2B_DIGEST_Marshal(&responseAuth,
&command->responseBuffer,
NULL);

if (!IS_ATTRIBUTE(s_attributes[i], TPMA_SESSION, continueSession))
    SessionFlush(s_sessionHandles[i]);

6.4.5.13 SessionRemoveAssociationToHandle()

This function deals with the case where an entity associated with an authorization is deleted during command processing. The primary use of this is to support UndefineSpaceSpecial().

void SessionRemoveAssociationToHandle(
    TPM_HANDLE handle
)
{
    UINT32 i;
    //
    for (i = 0; i < MAX_SESSION_NUM; i++)
    {
        if (s_associatedHandles[i] == handle)
        {
            s_associatedHandles[i] = TPM_RH_NULL;
        }
    }
}
7 Command Support Functions

7.1 Introduction

This clause contains support routines that are called by the command action code in TPM 2.0 Part 3. The functions are grouped by the command group that is supported by the functions.

7.2 Attestation Command Support (Attest_spt.c)

7.2.1 Includes

```c
#include "Tpm.h"
#include "Attest_spt_fp.h"
```

7.2.2 Functions

7.2.2.1 FillInAttestInfo()

Fill in common fields of TPMS_ATTEST structure.

```c
void FillInAttestInfo(
    TPMI_DH_OBJECT       signHandle,        // IN: handle of signing object
    TPM_TSIG_SCHEME     *scheme,            // IN/OUT: scheme to be used for signing
    TPM2B_DATA          *data,              // IN: qualifying data
    TPMS_ATTEST         *attest             // OUT: attest structure
)
{
    OBJECT *signObject = HandleToObject(signHandle);
    attest->magic = TPM_GENERATED_VALUE;
    if(signObject == NULL)
        if(the name for a null handle is TPM_RH_NULL)
            // This is defined because UINT32_TO_BYTE_ARRAY does a cast. If the
            // size of the cast is smaller than a constant, the compiler warns
            // about the truncation of a constant value.
            TPM_HANDLE nullHandle = TPM_RH_NULL;
            attest->qualifiedSigner.t.size = sizeof(TPM_HANDLE);
            UINT32_TO_BYTE_ARRAY(nullHandle, attest->qualifiedSigner.t.name);
    else
        attest->qualifiedSigner = signObject->qualifiedName;
    // current clock in plain text
    TimeFillInfo(&attest->clockInfo);
    // Firmware version in plain text
    attest->firmwareVersion = ((UINT64)gp.firmwareV1 << (sizeof(UINT32) * 8));
    attest->firmwareVersion += gp.firmwareV2;
```
// Check the hierarchy of sign object. For NULL sign handle, the hierarchy
// will be TPM_RH_NULL
if((signObject == NULL)
   || (!signObject->attributes.epsHierarchy
      && !signObject->attributes.ppsHierarchy))
{
    // For signing key that is not in platform or endorsement hierarchy,
    // obfuscate the reset, restart and firmware version information
    UINT64          obfuscation[2];
    CryptKDFa(CONTEXT_INTEGRITY_HASH_ALG, &gp.shProof.b, OBFUSCATE_STRING,
              &attest->qualifiedSigner.b, NULL, 128,
              (BYTE *)&obfuscation[0], NULL, FALSE);
    // Obfuscate data
    attest->firmwareVersion += obfuscation[0];
    attest->clockInfo.resetCount += (UINT32)(obfuscation[1] >> 32);
    attest->clockInfo.restartCount += (UINT32)obfuscation[1];
}

// External data
if(CryptIsSchemeAnonymous(scheme->scheme))
   attest->extraData.t.size = 0;
else
{
    // If we move the data to the attestation structure, then it is not
    // used in the signing operation except as part of the signed data
    attest->extraData = *data;
    data->t.size = 0;
}

7.2.2.2 SignAttestInfo()

Sign a TPMS_ATTEST structure. If signHandle is TPM_RH_NULL, a null signature is returned.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>signHandle references not a signing key</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>scheme is not compatible with signHandle type</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>digest generated for the given scheme is greater than the modulus of signHandle (for an RSA key); invalid commit status or failed to generate r value (for an ECC key)</td>
</tr>
</tbody>
</table>

TPM_RC
SignAttestInfo(

OBJECT              *signKey,       // IN: sign object
TPMT_SIG_SCHEME     *scheme,        // IN: sign scheme
TPMS_ATTEST         *certifyInfo,   // IN: the data to be signed
TPM2B_DATA          *qualifiedData, // IN: extra data for the signing
TPM2B_ATTEST        *attest,        // OUT: marshaled attest blob to be
TPMT_SIGNATURE      *signature      // OUT: signature
)
{
    BYTE                    *buffer;
    HASH_STATE              hashState;
    TPM2B_DIGEST           digest;
    TPM_RC                  result;

    // Marshal TPMS_ATTEST structure for hash
    buffer = attest->t.attestationData;
    attest->t.size = TPMS_ATTEST_Marshal(certifyInfo, &buffer, NULL);
    if(signKey == NULL)


7.2.2.3 IsSigningObject()

Checks to see if the object is OK for signing. This is here rather than in Object_spt.c because all the attestation commands use this file but not Object_spt.c.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>object may sign</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>object may not sign</td>
</tr>
</tbody>
</table>

```c
BOOL IsSigningObject(
    OBJECT *object   // IN:  
) {
    return ((object == NULL)
             || ((IS_ATTRIBUTE(object->publicArea.objectAttributes, TPMA_OBJECT, sign)
                 && object->publicArea.type != TPM_ALG_SYMCIPHER));
}
```
7.3  Context Management Command Support (Context_spt.c)

7.3.1  Includes

```c
#include "Tpm.h"
#include "Context_spt_fp.h"
```

7.3.2  Functions

7.3.2.1  ComputeContextProtectionKey()

This function retrieves the symmetric protection key for context encryption. It is used by
TPM2_ContextSave() and TPM2_ContextLoad() to create the symmetric encryption key and iv

```c
void ComputeContextProtectionKey(
    TPM2B_DATA sequence2B, handle2B;
)
{ ...
    // Get sequence value in 2B format
    sequence2B.t.size = sizeof(contextBlob->sequence);
    cAssert(sizeof(contextBlob->sequence) <= sizeof(sequence2B.t.buffer));
    MemoryCopy(sequence2B.t.buffer, &contextBlob->sequence,
               sizeof(contextBlob->sequence));

    // Get handle value in 2B format
    handle2B.t.size = sizeof(contextBlob->savedHandle);
    cAssert(sizeof(contextBlob->savedHandle) <= sizeof(handle2B.t.buffer));
    MemoryCopy(handle2B.t.buffer, &contextBlob->savedHandle,
               sizeof(contextBlob->savedHandle));

    // Get the symmetric encryption key size
    symKey->t.size = CONTEXT_ENCRYPT_KEY_BYTES;
    // Get the size of the IV for the algorithm
    iv->t.size = CryptGetSymmetricBlockSize(CONTEXT_ENCRYPT_ALG, symKeyBits);

    // KDFa to generate symmetric key and IV value
    CryptKDFa(CONTEXT_INTEGRITY_HASH_ALG, &proof->b, CONTEXT_KEY, &sequence2B.b,
              &handle2B.b, (symKey->t.size + iv->t.size) * 8, kdfResult, NULL,
              FALSE);

    // Copy part of the returned value as the key
    MemoryCopy(symKey->t.buffer, kdfResult, symKey->t.size);
}```
7.3.2.2 ComputeContextIntegrity()

Generate the integrity hash for a context. It is used by TPM2_ContextSave() to create an integrity hash and by TPM2_ContextLoad() to compare an integrity hash.

```c
void ComputeContextIntegrity(
    TPMS_CONTEXT *contextBlob, // IN: context blob
    TPM2B_DIGEST *integrity  // OUT: integrity
) {
    HMAC_STATE hmacState;
    TPM2B_PROOF *proof;
    UINT16 integritySize;

    // Get proof value
    proof = HierarchyGetProof(contextBlob->hierarchy);

    // Start HMAC
    integrity->t.size = CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG,
                                         &proof->b);

    // Compute integrity size at the beginning of context blob
    integritySize = sizeof(integrity->t.size) + integrity->t.size;

    // Adding total reset counter so that the context cannot be
    // used after a TPM Reset
    CryptDigestUpdateInt(&hmacState.hashState, sizeof(gp.totalResetCount),
                         gp.totalResetCount);

    // If this is a ST_CLEAR object, add the clear count
    // so that this context cannot be loaded after a TPM Restart
    if (contextBlob->savedHandle == 0x80000002) {
        CryptDigestUpdateInt(&hmacState.hashState, sizeof(gr.clearCount),
                             gr.clearCount);
    }

    // Adding sequence number to the HMAC to make sure that it doesn't
    // get changed
    CryptDigestUpdateInt(&hmacState.hashState, sizeof(contextBlob->sequence),
                         contextBlob->sequence);

    // Protect the handle
    CryptDigestUpdateInt(&hmacState.hashState, sizeof(contextBlob->savedHandle),
                         contextBlob->savedHandle);

    // Adding sensitive contextData, skip the leading integrity area
    CryptDigestUpdate(&hmacState.hashState,
                      contextBlob->contextBlob.t.size - integritySize,
                      contextBlob->contextBlob.t.buffer + integritySize);

    // Complete HMAC
    CryptHmacEnd2B(&hmacState, &integrity->b);
    return;
}
```
7.3.2.3 SequenceDataExport()

This function is used scan through the sequence object and either modify the hash state data for export (contextSave) or to import it into the internal format (contextLoad). This function should only be called after the sequence object has been copied to the context buffer (contextSave) or from the context buffer into the sequence object. The presumption is that the context buffer version of the data is the same size as the internal representation so nothing outside of the hash context area gets modified.

```c
void SequenceDataExport(
  HASH_OBJECT         *object,  // IN: an internal hash object
  HASH_OBJECT_BUFFER  *exportObject  // OUT: a sequence context in a buffer
)
{
  // If the hash object is not an event, then only one hash context is needed
  int count = (object->attributes.eventSeq) ? HASH_COUNT : 1;

  for(count--; count >= 0; count--)
  {
    HASH_STATE          *hash = &object->state.hashState[count];
    size_t               offset = (BYTE *)hash - (BYTE *)object;
    BYTE                *exportHash = &((BYTE *)exportObject)[offset];

    CryptHashExportState(hash, (EXPORT_HASH_STATE *)exportHash);
  }
}
```

7.3.2.4 SequenceDataImport()

This function is used scan through the sequence object and either modify the hash state data for export (contextSave) or to import it into the internal format (contextLoad). This function should only be called after the sequence object has been copied to the context buffer (contextSave) or from the context buffer into the sequence object. The presumption is that the context buffer version of the data is the same size as the internal representation so nothing outside of the hash context area gets modified.

```c
void SequenceDataImport(
  HASH_OBJECT         *object,  // IN/OUT: an internal hash object
  HASH_OBJECT_BUFFER  *exportObject  // IN/OUT: a sequence context in a buffer
)
{
  // If the hash object is not an event, then only one hash context is needed
  int count = (object->attributes.eventSeq) ? HASH_COUNT : 1;

  for(count--; count >= 0; count--)
  {
    HASH_STATE          *hash = &object->state.hashState[count];
    size_t               offset = (BYTE *)hash - (BYTE *)object;
    BYTE                *importHash = &((BYTE *)exportObject)[offset];

    CryptHashImportState(hash, (EXPORT_HASH_STATE *)importHash);
  }
}
```
7.4 Policy Command Support (Policy_spt.c)

7.4.1 Includes

```c
#include "Tpm.h"
#include "Policy_spt_fp.h"
#include "PolicySigned_fp.h"
#include "PolicySecret_fp.h"
#include "PolicyTicket_fp.h"
```

7.4.2 Functions

7.4.2.1 PolicyParameterChecks()

This function validates the common parameters of TPM2_PolicySigned() and TPM2_PolicySecret(). The common parameters are nonceTPM, expiration, and cpHashA.

```c
TPM_RC PolicyParameterChecks(
    SESSION         *session,
    UINT64           authTimeout,
    TPM2B_DIGEST    *cpHashA,
    TPM2B_NONCE     *nonce,
    TPM_RC          blameNonce,
    TPM_RC          blameCpHash,
    TPM_RC          blameExpiration
)
{
    // Validate that input nonceTPM is correct if present
    if(nonce != NULL && nonce->t.size != 0)
    {
        if(!MemoryEqual2B(nonce->b, session->nonceTPM.b))
            return TPM_RC_NONCE + blameNonce;
    }
    // If authTimeout is set (expiration != 0... )
    if(authTimeout != 0)
    {
        // Validate input expiration.
        // Cannot compare time if clock stop advancing. A TPM_RC_NV_UNAVAILABLE
        // or TPM_RC_NV_RATE error may be returned here.
        RETURN_IF_NV_IS_NOT_AVAILABLE;
        // if the time has already passed or the time epoch has changed then the
        // time value is no longer good.
        if((authTimeout < g_time)
            || (session->epoch != g_timeEpoch))
            return TPM_RC_EXPIRED + blameExpiration;
    }
    // If the cpHash is present, then check it
    if(cpHashA != NULL && cpHashA->t.size != 0)
    {
        // The cpHash input has to have the correct size
        if(cpHashA->t.size != session->u2.policyDigest.t.size)
            return TPM_RC_SIZE + blameCpHash;
        // If the cpHash has already been set, then this input value
        // must match the current value.
        if( MemoryEqual2B(cpHashA->b, session->u1.cpHash.b) )
            return TPM_RC_CPHASH;
    }
}
7.4.2.2 PolicyContextUpdate()

Update policy hash Update the policyDigest in policy session by extending policyRef and objectName to it. This will also update the cpHash if it is present.

```c
void PolicyContextUpdate(
    TPM_CC commandCode, // IN: command code
    TPM2B_NAME *name, // IN: name of entity
    TPM2B_NONCE *ref, // IN: the reference data
    TPM2B_DIGEST *cpHash, // IN: the cpHash (optional)
    UINT64 policyTimeout, // IN: the timeout value for the policy
    SESSION *session // IN/OUT: policy session to be updated
)
{
    HASH_STATE hashState;
    // Start hash
    CryptHashStart(&hashState, session->authHashAlg);
    // policyDigest size should always be the digest size of session hash algorithm.
    pAssert(session->u2.policyDigest.t.size == CryptHashGetDigestSize(session->authHashAlg));
    // add old digest
    CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);
    // add commandCode
    CryptDigestUpdateInt(&hashState, sizeof(commandCode), commandCode);
    // add name if applicable
    if (name != NULL)
        CryptDigestUpdate2B(&hashState, &name->b);
    // Complete the digest and get the results
    CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);
    // If the policy reference is not null, do a second update to the digest.
    if (ref != NULL)
    {
        // Start second hash computation
        CryptHashStart(&hashState, session->authHashAlg);
        // add policyDigest
        CryptDigestUpdate2B(&hashState, &session->u2.policyDigest.b);
        // add policyRef
        CryptDigestUpdate2B(&hashState, &ref->b);
        // Complete second digest
        CryptHashEnd2B(&hashState, &session->u2.policyDigest.b);
    }
    // Deal with the cpHash. If the cpHash value is present
    // then it would have already been checked to make sure that
    // it is compatible with the current value so all we need
    // to do here is copy it and set the isCpHashDefined attribute
    if (cpHash != NULL && cpHash->t.size != 0)
    {
        session->u1.cpHash = *cpHash;
        session->attributes.isCpHashDefined = SET;
    }
```

return TPM_RC_SUCCESS;
108  }
109
110  // update the timeout if it is specified
111  if(policyTimeout != 0)
112  {
113  // If the timeout has not been set, then set it to the new value
114  // than the current timeout then set it to the new value
115  if(session->timeout == 0 || session->timeout > policyTimeout)
116    session->timeout = policyTimeout;
117  }
118  return;
119 }

7.4.2.3  ComputeAuthTimeout()

This function is used to determine what the authorization timeout value for the session should be.

120  UINT64
121  ComputeAuthTimeout(  
122    SESSION         *session,  // IN: the session containing the time
123    // values  
124    INT32            expiration, // IN: either the number of seconds from
125    // the start of the session or the
126    // time in g_timer;
127    TPM2B_NONCE     *nonce      // IN: indicator of the time base
128  )
129  {
130  UINT64           policyTime;
131  // If no expiration, policy time is 0
132  if(expiration == 0)
133    policyTime = 0;
134  else
135  {
136  if(expiration < 0)
137    expiration = -expiration;
138  if(nonce->t.size == 0)
139    // The input time is absolute Time (not Clock), but it is expressed
140    // in seconds. To make sure that we don’t time out too early, take the
141    // current value of milliseconds in g_time and add that to the input
142    // seconds value.
143    policyTime = (((UINT64)expiration) * 1000) + g_time % 1000;
144  else
145    // The policy timeout is the absolute value of the expiration in seconds
146    // added to the start time of the policy.
147    policyTime = session->startTime + (((UINT64)expiration) * 1000);
148  }
149  return policyTime;
150 }

7.4.2.4  PolicyDigestClear()

Function to reset the policyDigest of a session

152  void
153  PolicyDigestClear(  
154    SESSION         *session
155  )
156  {
157    session->u2.policyDigest.t.size = CryptHashGetDigestSize(session->authHashAlg);
158    MemorySet(session->u2.policyDigest.t.buffer, 0,
159        session->u2.policyDigest.t.size);
160  }

    }
BOOL PolicySptCheckCondition(  
   TPM EO operation,  
   BYTE *opA,  
   BYTE *opB,  
   UINT16 size  
)
{
   // Arithmetic Comparison
   switch(operation)
   {
      case TPM EO_EQ:
         // compare A = B
         return (UnsignedCompareB(size, opA, size, opB) == 0);
         break;
      case TPM EO_NEQ:
         // compare A != B
         return (UnsignedCompareB(size, opA, size, opB) != 0);
         break;
      case TPM EO_SIGNED_GT:
         // compare A > B signed
         return (SignedCompareB(size, opA, size, opB) > 0);
         break;
      case TPM EO_UNSIGNED_GT:
         // compare A > B unsigned
         return (UnsignedCompareB(size, opA, size, opB) > 0);
         break;
      case TPM EO_SIGNED_LT:
         // compare A < B signed
         return (SignedCompareB(size, opA, size, opB) < 0);
         break;
      case TPM EO_UNSIGNED_LT:
         // compare A < B unsigned
         return (UnsignedCompareB(size, opA, size, opB) < 0);
         break;
      case TPM EO_SIGNED_GE:
         // compare A >= B signed
         return (SignedCompareB(size, opA, size, opB) >= 0);
         break;
      case TPM EO_UNSIGNED_GE:
         // compare A >= B unsigned
         return (UnsignedCompareB(size, opA, size, opB) >= 0);
         break;
      case TPM EO_SIGNED_LE:
         // compare A <= B signed
         return (SignedCompareB(size, opA, size, opB) <= 0);
         break;
      case TPM EO_UNSIGNED_LE:
         // compare A <= B unsigned
         return (UnsignedCompareB(size, opA, size, opB) <= 0);
         break;
      case TPM EO_BITSET:
         // All bits SET in B are SET in A. ((A&B)=B)
         {
            UINT32 i;
            for(i = 0; i < size; i++)
               if((opA[i] & opB[i]) != opB[i])
                  return FALSE;
         }
         break;
      case TPM EO_BITCLEAR:
         // All bits SET in B are CLEAR in A. ((A&B)=0)
         {
            UINT32 i;
            for(i = 0; i < size; i++)
               if((opA[i] & opB[i]) != 0)
return FALSE;
break;

default:
FAIL(FATAL_ERROR_INTERNAL);
break;
}

return TRUE;
7.5 NV Command Support (NV_spt.c)

7.5.1 Includes

```
#include "Tpm.h"
#include "NV_spt_fp.h"
```

7.5.2 Functions

7.5.2.1 NvReadAccessChecks()

Common routine for validating a read Used by TPM2_NV_Read(), TPM2_NV_ReadLock() and TPM2_PolicyNV()

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>authHandle is not allowed to authorize read of the index</td>
</tr>
<tr>
<td>TPM_RC_NV_LOCKED</td>
<td>Read locked</td>
</tr>
<tr>
<td>TPM_RC_NV_UNINITIALIZED</td>
<td>Try to read an uninitialized index</td>
</tr>
</tbody>
</table>

```c
TPM_RC
NvReadAccessChecks(
    TPM_HANDLE authHandle, // IN: the handle that provided the
    TPM_HANDLE nvHandle,   // IN: the handle of the NV index to be read
    TPMA_NV attributes // IN: the attributes of 'nvHandle'
)
{
    // If data is read locked, returns an error
    if(IS_ATTRIBUTE(attributes, TPMA_NV, READLOCKED))
        return TPM_RC_NV_LOCKED;
    // If the authorization was provided by the owner or platform, then check
    // that the attributes allow the read. If the authorization handle
    // is the same as the index, then the checks were made when the authorization
    // was checked..
    if(authHandle == TPM_RH_OWNER)
        // If Owner provided authorization then ONWERWRITE must be SET
        if(!IS_ATTRIBUTE(attributes, TPMA_NV, OWNERREAD))
            return TPM_RC_NV_AUTHORIZATION;
    } else if(authHandle == TPM_RH_PLATFORM)
        // If Platform provided authorization then PWRITE must be SET
        if(!IS_ATTRIBUTE(attributes, TPMA_NV, PPREAD))
            return TPM_RC_NV_AUTHORIZATION;
    
    // If neither Owner nor Platform provided authorization, make sure that it was
    // provided by this index.
    else if(authHandle != nvHandle)
        return TPM_RC_NV_AUTHORIZATION;
    // If the index has not been written, then the value cannot be read
    // NOTE: This has to come after other access checks to make sure that
    // the proper authorization is given to TPM2_NV_ReadLock()
    if(!IS_ATTRIBUTE(attributes, TPMA_NV, WRITTEN))
        return TPM_RC_NV_UNINITIALIZED;
    return TPM_RC_SUCCESS;
}
```
### 7.5.2.2 NvWriteAccessChecks()

Common routine for validating a write Used by TPM2_NV_Write(), TPM2_NV_Increment(), TPM2_SetBits(), and TPM2_NV_WriteLock()

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>Authorization fails</td>
</tr>
<tr>
<td>TPM_RC_NV_LOCKED</td>
<td>Write locked</td>
</tr>
</tbody>
</table>

```c
TPM_RC
NvWriteAccessChecks(
    TPM_HANDLE       authHandle, // IN: the handle that provided the
                      // authorization
    TPM_HANDLE       nvHandle, // IN: the handle of the NV index to be written
    TPMA_NV          attributes // IN: the attributes of 'nvHandle'
) {
    // If data is write locked, returns an error
    if(IS_ATTRIBUTE(attributes, TPMA_NV, WRITELOCKED))
        return TPM_RC_NV_LOCKED;
    // If the authorization was provided by the owner or platform, then check
    // that the attributes allow the write. If the authorization handle
    // is the same as the index, then the checks were made when the authorization
    // was checked.
    if(authHandle == TPM_RH_OWNER)
        { // If Owner provided authorization then ONWERWRITE must be SET
            if(!IS_ATTRIBUTE(attributes, TPMA_NV, OWNERWRITE))
                return TPM_RC_NV_AUTHORIZATION;
        }
    else if(authHandle == TPM_RH_PLATFORM)
        { // If Platform provided authorization then PPWRITE must be SET
            if(!IS_ATTRIBUTE(attributes, TPMA_NV, PPWRITE))
                return TPM_RC_NV_AUTHORIZATION;
        }
    // If neither Owner nor Platform provided authorization, make sure that it was
    // provided by this index.
    else if(authHandle != nvHandle)
        return TPM_RC_NV_AUTHORIZATION;
    return TPM_RC_SUCCESS;
}
```

### 7.5.2.3 NvClearOrderly()

This function is used to cause gp.orderlyState to be cleared to the non-orderly state.

```c
TPM_RC
NvClearOrderly(
    void
) {
    if(gp.orderlyState < SU_DA_USED_VALUE)
        RETURN_IF_NV_IS_NOT_AVAILABLE;
    g_clearOrderly = TRUE;
    return TPM_RC_SUCCESS;
}
```
7.5.2.4 NvIsPinPassIndex()

Function to check to see if an NV index is a PIN Pass Index

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>is pin pass</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>is not pin pass</td>
</tr>
</tbody>
</table>

```c
BOOL NvIsPinPassIndex(
    TPM_HANDLE          index       // IN: Handle to check
) {
    if (HandleGetType(index) == TPM_HT_NV_INDEX)
    {
        NV_INDEX                *nvIndex = NvGetIndexInfo(index, NULL);
        return IsNvPinPassIndex(nvIndex->publicArea.attributes);
    }
    return FALSE;
}
```
7.6 Object Command Support (Object_spt.c)

7.6.1 Includes

```c
#include "Tpm.h"
#include "Object_spt_fp.h"
```

7.6.2 Local Functions

7.6.2.1 GetIV2BSize()

Get the size of TPM2B_IV in canonical form that will be append to the start of the sensitive data. It includes both size of size field and size of iv data

```c
static UINT16
GetIV2BSize(
OBJECT              *protector          // IN: the protector handle
)
{
    TPM_ALG_ID    symAlg;
    UINT16        keyBits;

    // Determine the symmetric algorithm and size of key
    if(protector == NULL)
    {
        // Use the context encryption algorithm and key size
        symAlg = CONTEXT_ENCRYPT_ALG;
        keyBits = CONTEXT_ENCRYPT_KEY_BITS;
    }
    else
    {
        symAlg = protector->publicArea.parameters.asymDetail.symmetric.algorithm;
        keyBits = protector->publicArea.parameters.asymDetail.symmetric.keyBits.sym;
    }

    // The IV size is a UINT16 size field plus the block size of the symmetric
    // algorithm
    return sizeof(UINT16) + CryptGetSymmetricBlockSize(symAlg, keyBits);
}
```

7.6.2.2 ComputeProtectionKeyParms()

This function retrieves the symmetric protection key parameters for the sensitive data. The parameters retrieved from this function include encryption algorithm, key size in bit, and a TPM2B_SYM_KEY containing the key material as well as the key size in bytes. This function is used for any action that requires encrypting or decrypting of the sensitive area of an object or a credential blob.

```c
static void
ComputeProtectionKeyParms(
OBJECT              *protector,         // IN: the protector object
TPM_ALG_ID          hashAlg,          // IN: hash algorithm for KDFa
TPM2B              *name,             // IN: name of the object
TPM2B              *seedIn,           // IN: optional seed for duplication blob.
TPM_ALG_ID          *symAlg,          // OUT: the symmetric algorithm
UINT16              *keyBits,         // OUT: the symmetric key size in bits
TPM2B_SYM_KEY       *symKey           // OUT: the symmetric key
)
```
40 {  
41    const TPM2B *seed = seedIn;
42
43    // Determine the algorithms for the KDF and the encryption/decryption
44    // For TPM RH.NULL, using context settings
45    if(protector == NULL)
46    {
47        // Use the context encryption algorithm and key size
48        *symAlg = CONTEXT_ENCRYPT_ALG;
49        symKey->t.size = CONTEXT_ENCRYPT_KEY_BYTES;
50        *keyBits = CONTEXT_ENCRYPT_KEY_BITS;
51    }
52    else
53    {
54        TPMT_SYM_DEF_OBJECT *symDef;
55        symDef = &protector->publicArea.parameters.asymDetail.symmetric;
56        *symAlg = symDef->algorithm;
57        *keyBits = symDef->keyBits.sym;
58        symKey->t.size = (*keyBits + 7) / 8;
59    }
60    // Get seed for KDF
61    if(seed == NULL)
62    {
63        seed = GetSeedForKDF(protector);
64    }  
65    // KDFa to generate symmetric key and IV value
66    CryptKDFa(hashAlg, seed, STORAGE_KEY, name, NULL,
67             symKey->t.size * 8, symKey->t.buffer, NULL, FALSE);
68    return;
69 }

7.6.2.3 ComputeOuterIntegrity()

The sensitive area parameter is a buffer that holds a space for the integrity value and the marshaled sensitive area. The caller should skip over the area set aside for the integrity value and compute the hash of the remainder of the object. The size field of sensitive is in unmarshaled form and the sensitive area contents is an array of bytes.

static void ComputeOuterIntegrity(
    TPM2B *name,  // IN: the name of the object
    OBJECT *protector,  // IN: the object that
    TPMI_ALG_HASH hashAlg,  // IN: algorithm to use for integrity
    TPM2B *seedIn,  // IN: an external seed may be provided for
    TPM2B_DIGEST *integrity,  // OUT: integrity
)
{
    HMAC_STATE hmacState;
    TPM2B_DIGEST hmacKey;
    const TPM2B *seed = seedIn;

    // Get seed for KDF
    if(seed == NULL)
    {
        seed = GetSeedForKDF(protector);
    }
    // Determine the HMAC key bits
    hmacKey.t.size = CryptHashGetDigestSize(hashAlg);
    // KDFa to generate HMAC key
96   CryptKDFa(hashAlg, seed, INTEGRITY_KEY, NULL, NULL, 
97       hmacKey.t.size * 8, hmacKey.t.buffer, NULL, FALSE);
98   // Start HMAC and get the size of the digest which will become the integrity
99   integrity->t.size = CryptHmacStart2B(&hmacState, hashAlg, &hmacKey.b);
100  // Adding the marshaled sensitive area to the integrity value
101  CryptDigestUpdate(&hmacState.hashState, sensitiveSize, sensitiveData);
102  // Adding name
103  CryptDigestUpdate2B(&hmacState.hashState, name);
104  // Compute HMAC
105  CryptHmacEnd2B(&hmacState, &integrity->b);
106  return;
107 }

7.6.2.4  ComputeInnerIntegrity()

This function computes the integrity of an inner wrap

112  static void
113  ComputeInnerIntegrity(
114     TPM_ALG_ID       hashAlg,  // IN: hash algorithm for inner wrap
115     TPM2B           *name,    // IN: the name of the object
116     UINT16           dataSize,  // IN: the size of sensitive data
117     BYTE            *sensitiveData,  // IN: sensitive data
118     TPM2B_DIGEST    *integrity // OUT: inner integrity
119 )
120 {
121     HASH_STATE      hashState;
122     // Start hash and get the size of the digest which will become the integrity
123     integrity->t.size = CryptHashStart(&hashState, hashAlg);
124     // Adding the marshaled sensitive area to the integrity value
125     CryptDigestUpdate(&hashState, dataSize, sensitiveData);
126     // Adding name
127     CryptDigestUpdate2B(&hashState, name);
128     // Compute hash
129     CryptHashEnd2B(&hashState, &integrity->b);
130     return;
131 }

7.6.2.5  ProduceInnerIntegrity()

This function produces an inner integrity for regular private, credential or duplication blob. It requires the sensitive data being marshaled to the innerBuffer, with the leading bytes reserved for integrity hash. It assumes the sensitive data starts at address (innerBuffer + integrity size). This function integrity at the beginning of the inner buffer. It returns the total size of buffer with the inner wrap

137  static UINT16
138  ProduceInnerIntegrity(
139     TPM2B           *name,  // IN: the name of the object
140     TPM_ALG_ID       hashAlg,  // IN: hash algorithm for inner wrap
141     UINT16           dataSize,  // IN: the size of sensitive data, excluding the
142                     // leading integrity buffer size
143     BYTE            *innerBuffer  // IN/OUT: inner buffer with sensitive data in
144                     // it. At input, the leading bytes of this
145     // buffer is reserved for integrity
146 }
147 {
148     BYTE *sensitiveData; // pointer to the sensitive data
149     TPM2B_DIGEST integrity;
150     UINT16 integritySize;
151     BYTE *buffer; // Auxiliary buffer pointer
152     // sensitiveData points to the beginning of sensitive data in innerBuffer
153     integritySize = sizeof(UINT16) + CryptHashGetDigestSize(hashAlg);
154     sensitiveData = innerBuffer + integritySize;
155     ComputeInnerIntegrity(hashAlg, name, dataSize, sensitiveData, &integrity);
156     // Add integrity at the beginning of inner buffer
157     buffer = innerBuffer;
158     TPM2B_DIGEST_Marshal(&integrity, &buffer, NULL);
159     return dataSize + integritySize;
160 }

7.6.2.6 CheckInnerIntegrity()

This function check integrity of inner blob

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_INTEGRITY</td>
<td>if the outer blob integrity is bad</td>
</tr>
<tr>
<td>unmarshal errors</td>
<td>unmarshal errors while unmarshaling integrity</td>
</tr>
</tbody>
</table>

165    static TPM_RC
166    CheckInnerIntegrity(
167        TPM2B *name, // IN: the name of the object
168        TPM_ALG_ID hashAlg, // IN: hash algorithm for inner wrap
169        UINT16 dataSize, // IN: the size of sensitive data, including the
170        // leading integrity buffer size
171        BYTE *innerBuffer // IN/OUT: inner buffer with sensitive data in
172        // it
173    )
174 {
175     TPM_RC result;
176     TPM2B_DIGEST integrity;
177     TPM2B_DIGEST integrityToCompare;
178     BYTE *buffer; // Auxiliary buffer pointer
179     INT32 size;
180     // Unmarshal integrity
181     buffer = innerBuffer;
182     size = (INT32)dataSize;
183     result = TPM2B_DIGEST_Unmarshal(&integrity, &buffer, &size);
184     if(result == TPM_RC_SUCCESS)
185     {
186         // Compute integrity to compare
187         ComputeInnerIntegrity(hashAlg, name, (UINT16)size, buffer,
188                     &integrityToCompare);
189         // Compare outer blob integrity
190         if(!MemoryEqual2B(&integrity.b, &integrityToCompare.b))
191         result = TPM_RC_INTEGRITY;
192     }
193     return result;
194 }
7.6.3 Public Functions

7.6.3.1 AdjustAuthSize()

This function will validate that the input authValue is no larger than the digestSize for the nameAlg. It will then pad with zeros to the size of the digest.

```c
BOOL AdjustAuthSize(
    TPM2B_AUTH *auth, // IN/OUT: value to adjust
    TPMI_ALG_HASH nameAlg // IN:
)
{
    UINT16               digestSize;
    // If there is no nameAlg, then this is a LoadExternal and the authVale can
    // be any size up to the maximum allowed by the
    digestSize = (nameAlg == TPM_ALG_NULL) ? sizeof(TPMU_HA)
               : CryptHashGetDigestSize(nameAlg);
    if(digestSize < MemoryRemoveTrailingZeros(auth))
        return FALSE;
    else if(digestSize > auth->t.size)
        MemoryPad2B(&auth->b, digestSize);
    auth->t.size = digestSize;
    return TRUE;
}
```

7.6.3.2 AreAttributesForParent()

This function is called by create, load, and import functions.

NOTE: The isParent attribute is SET when an object is loaded and it has attributes that are suitable for a parent object.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>properties are those of a parent</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>properties are not those of a parent</td>
</tr>
</tbody>
</table>

```c
BOOL ObjectIsParent(  
    OBJECT          *parentObject // IN: parent handle
)
{
    return parentObject->attributes.isParent;
}
```

7.6.3.3 CreateChecks()

Attribute checks that are unique to creation.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>sensitiveDataOrigin is not consistent with the object type</td>
</tr>
<tr>
<td>other</td>
<td>returns from PublicAttributesValidation()</td>
</tr>
</tbody>
</table>

```c
TPM_RC CreateChecks(
```
OBJECT *parentObject,
TPMTP_PUBLIC *publicArea,
UINT16 sensitiveDataSize
}

TPMA_OBJECT attributes = publicArea->objectAttributes;
TPM_RC result = TPM_RC_SUCCESS;

// If the caller indicates that they have provided the data, then make sure that
// they have provided some data.
if((!IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
   && (sensitiveDataSize == 0))
   return TPM_RC_ATTRIBUTES;

// For an ordinary object, data can only be provided when sensitiveDataOrigin
// is CLEAR
if((parentObject != NULL)
   && (IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
   && (sensitiveDataSize != 0))
   return TPM_RC_ATTRIBUTES;

switch(publicArea->type)
{
    case ALG_KEYEDHASH_VALUE:
    // if this is a data object (sign == decrypt == CLEAR) then the
    // TPM cannot be the data source.
    if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
       && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt)
       && IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
       result = TPM_RC_ATTRIBUTES;
    // comment out the next line in order to prevent a fixedTPM derivation
    //            break;
    case ALG_SYMCIPHER_VALUE:
    // A restricted key symmetric key (SYMCIPHER and KEYEDHASH)
    // must have sensitiveDataOrigin SET unless it has fixedParent and
    // fixedTPM CLEAR.
    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted))
       if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
           if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent)
               || IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM))
               result = TPM_RC_ATTRIBUTES;
       break;
    default: // Asymmetric keys cannot have the sensitive portion provided
        if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, sensitiveDataOrigin))
           result = TPM_RC_ATTRIBUTES;
        break;
}
if(TPM_RC_SUCCESS == result)
{
    result = PublicAttributesValidation(parentObject, publicArea);
}

return result;

7.6.3.4 SchemeChecks

This function is called by TPM2_LoadExternal() and PublicAttributesValidation(). This function validates the schemes in the public area of an object.
<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HASH</td>
<td>non-duplicable storage key and its parent have different name algorithm</td>
</tr>
<tr>
<td>TPM_RC_KDF</td>
<td>incorrect KDF specified for decrypting keyed hash object</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>invalid key size values in an asymmetric key public area</td>
</tr>
<tr>
<td>TPM_RCS_SCHEME</td>
<td>inconsistent attributes decrypt, sign, restricted and key's scheme ID; or hash algorithm is inconsistent with the scheme ID for keyed hash object</td>
</tr>
<tr>
<td>TPM_RC_SYMMETRIC</td>
<td>a storage key with no symmetric algorithm specified; or non-storage key with symmetric algorithm different from ALG_NULL</td>
</tr>
</tbody>
</table>

```c
TPM_RC
SchemeChecks(
    OBJECT *parentObject,  // IN: parent (null if primary seed)
    TPMT_PUBLIC *publicArea  // IN: public area of the object
)
{
    TPMT_SYM_DEF_OBJECT *symAlgs = NULL;
    TPM_ALG_ID scheme = TPM_ALG_NULL;
    TPMA_OBJECT attributes = publicArea->objectAttributes;
    TPMU_PUBLIC_PARMS *parms = &publicArea->parameters;

    // switch(publicArea->type)
    {
        case ALG_SYMCIPHER_VALUE:
            symAlgs = &parms->symDetail.sym;
            // If this is a decrypt key, then only the block cipher modes (not
            // SMAC) are valid. TPM_ALG_NULL is OK too. If this is a 'sign' key,
            // then any mode that got through the unmarshaling is OK.
            if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt)
                && !CryptSymModeIsValid(symAlgs->mode.sym, TRUE))
                return TPM_RCS_SCHEME;
            break;

        case ALG_KEYEDHASH_VALUE:
            scheme = parms->keyedHashDetail.scheme.scheme;
            // if both sign and decrypt
            if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
                == IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
            {
                // if both sign and decrypt are set or clear, then need
                // ALG_NULL as scheme
                if(scheme != TPM_ALG_NULL)
                    return TPM_RCS_SCHEME;
            }
            else if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
                && scheme != TPM_ALG_HMAC)
                return TPM_RCS_SCHEME;
            else if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
            {
                if(scheme != TPM_ALG_XOR)
                    return TPM_RCS_SCHEME;
                // If this is a derivation parent, then the KDF needs to be
                // SP800-108 for this implementation. This is the only derivation
                // supported by this implementation. Other implementations could
                // support additional schemes. There is no default.
                if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted))
                    {
                        if(parms->keyedHashDetail.scheme.details.xor.kdf
                            != TPM_ALG_KDF1_SP800_108)
                            return TPM_RCS_SCHEME;
                        // Must select a digest.
                    }
    ```
if(CryptHashGetDigestSize(
    parms->keyedHashDetail.scheme.details.xor.hashAlg) == 0)
    return TPM_RCS_HASH;

break;

default: // handling for asymmetric
    scheme = parms->asymDetail.scheme.scheme;
symAlgs = &parms->asymDetail.symmetric;
    // if the key is both sign and decrypt, then the scheme must be
    // ALG_NULL because there is no way to specify both a sign and a
    // decrypt scheme in the key.
    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
        == IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
        {
            // scheme must be TPM_ALG_NULL
            if(scheme != TPM_ALG_NULL)
                return TPM_RCS_SCHEME;
        }
    else if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign))
        {
            // If this is a signing key, see if it has a signing scheme
            if(CryptIsAsymSignScheme(publicArea->type, scheme))
                {
                    // if proper signing scheme then it needs a proper hash
                    if(parms->asymDetail.scheme.details.anySig.hashAlg
                        == TPM_ALG_NULL)
                        return TPM_RCS_SCHEME;
                }
            else
                {
                    // signing key that does not have a proper signing scheme.
                    // This is OK if the key is not restricted and its scheme
                    // is TPM_ALG_NULL
                    if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted)
                        || scheme != TPM_ALG_NULL)
                        return TPM_RCS_SCHEME;
                }
        }
    else if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
        {
            if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted))
                {
                    // for a restricted decryption key (a parent), scheme
                    // is required to be TPM_ALG_NULL
                    if(scheme != TPM_ALG_NULL)
                        return TPM_RCS_SCHEME;
                }
            else
                {
                    // For an unrestricted decryption key, the scheme has to
                    // be a valid scheme or TPM_ALG_NULL
                    if(scheme != TPM_ALG_NULL &&
                        !CryptIsAsymDecryptScheme(publicArea->type, scheme))
                        return TPM_RCS_SCHEME;
                }
        }
    if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted)
        || !IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
        {
            // For an asymmetric key that is not a parent, the symmetric
            // algorithms must be TPM_ALG_NULL
            if(symAlgs->algorithm != TPM_ALG_NULL)
                return TPM_RCS_SYMMETRIC;
        }

    // Special checks for an ECC key
#if ALG_ECC
    if (publicArea->type == TPM_ALG_ECC)
    {
        TPM_ECC_CURVE curveID;
        const TPMT_ECC_SCHEME *curveScheme;

        curveID = publicArea->parameters.eccDetail.curveID;
        curveScheme = CryptGetCurveSignScheme(curveID);
        // The curveId must be valid or the unmarshaling is busted.
        pAssert(curveScheme != NULL);

        // If the curveID requires a specific scheme, then the key must
        // select the same scheme
        if (curveScheme->scheme != TPM_ALG_NULL)
        {
            TPMS_ECC_PARMS *ecc = &publicArea->parameters.eccDetail;
            if (scheme != curveScheme->scheme)
            {
                return TPM_RCS_SCHEME;
            }
            // The scheme can allow any hash, or not...
            if (curveScheme->details.anySig.hashAlg != TPM_ALG_NULL
                && (ecc->scheme.details.anySig.hashAlg
                    != curveScheme->details.anySig.hashAlg))
            {
                return TPM_RCS_SCHEME;
            }
            // For now, the KDF must be TPM_ALG_NULL
            if (publicArea->parameters.eccDetail.kdf.scheme != TPM_ALG_NULL)
            {
                return TPM_RCS_KDF;
            }
        }
    }  
#endif

break;

// If this is a restricted decryption key with symmetric algorithms, then it
// is an ordinary parent (not a derivation parent). It needs to specific
// symmetric algorithms other than TPM_ALG_NULL
if (symAlgs != NULL
    && IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted)
    && IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
{
    if (symAlgs->algorithm == TPM_ALG_NULL)
    {
        return TPM_RCS_SYMMETRIC;
    }  
#if 0
    // This next check is under investigation. Need to see if it will break Windows
    // before it is enabled. If it does not, then it should be default because a
    // the mode used with a parent is always CFB and Part 2 indicates as much.
    if (symAlgs->mode.sym != TPM_ALG_CFB)
    {
        return TPM_RCS_MODE;
    }
#endif

    // If this parent is not duplicable, then the symmetric algorithms
    // (encryption and hash) must match those of its parent
    if (IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent)
        && (parentObject != NULL))
    {
        if (publicArea->nameAlg != parentObject->publicArea.nameAlg)
            return TPM_RCS_HASH;
        if (!MemoryEqual(symAlgs, &parentObject->publicArea.parameters,
            sizeof(TPMT_SYM_DEF_OBJECT)))
            return TPM_RCS_SYMMETRIC;
    }
}
return TPM_RC_SUCCESS;
7.6.3.5 PublicAttributesValidation()

This function validates the values in the public area of an object. This function is used in the processing of TPM2_Create(), TPM2_CreatePrimary(), TPM2_CreateLoaded(), TPM2_Load(), TPM2_Import(), and TPM2_LoadExternal(). For TPM2_Import() this is only used if the new parent has fixedTPM SET. For TPM2_LoadExternal(), this is not used for a public-only key.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>fixedTPM, fixedParent, or encryptedDuplication attributes are inconsistent between themselves or with those of the parent object; inconsistent restricted, decrypt and sign attributes; attempt to inject sensitive data for an asymmetric key; attempt to create a symmetric cipher key that is not a decryption key.</td>
</tr>
<tr>
<td>TPM_RC_HASH</td>
<td>nameAlg is TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>authPolicy.size does not match digest size of the name algorithm in publicArea</td>
</tr>
<tr>
<td>other</td>
<td>returns from SchemeChecks()</td>
</tr>
</tbody>
</table>

```c
TPM_RC
PublicAttributesValidation(
    OBJECT *parentObject, // IN: input parent object
    TPM2_PUBLIC *publicArea // IN: public area of the object
)
{
    TPMA_OBJECT attributes = publicArea->objectAttributes;
    TPMA_OBJECT parentAttributes = {0};
    //
    if (parentObject != NULL)
    {
        parentAttributes = parentObject->publicArea.objectAttributes;
        if (publicArea->nameAlg == TPM_ALG_NULL)
            return TPM_RC_HASH;

        // If there is an authPolicy, it needs to be the size of the digest produced
        // by the nameAlg of the object
        if ((publicArea->authPolicy.t.size != 0
             && (publicArea->authPolicy.t.size != CryptHashGetDigestSize(publicArea->nameAlg))))
            return TPM_RC_SIZE;

        // If the parent is fixedTPM (including a Primary Object) the object must have
        // the same value for fixedTPM and fixedParent
        if (parentObject == NULL
            || IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, fixedTPM))
            return TPM_RC_ATTRIBUTES;

        if (IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent)
            != IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM))
            return TPM_RC_ATTRIBUTES;
    }

    else
    {
        // The parent is not fixedTPM so the object can't be fixedTPM
        if (IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM))
            return TPM_RC_ATTRIBUTES;
    }

    // See if sign and decrypt are the same
    if (IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign)
        == IS_ATTRIBUTE(attributes, TPMA_OBJECT, decrypt))
    {
        // a restricted key cannot have both SET or both CLEAR
        if (IS_ATTRIBUTE(attributes, TPMA_OBJECT, restricted))
            return TPM_RC_ATTRIBUTES;

        // only a data object may have both sign and decrypt CLEAR
        // BTW, since we know that decrypt==sign, no need to check both
```
if(publicArea->type != TPM_ALG_KEYEDHASH
   && !IS_ATTRIBUTE(attributes, TPMA_OBJECT, sign))
   return TPM_RC_ATTRIBUTES;

// If the object can't be duplicated (directly or indirectly) then there
// is no justification for having encryptedDuplication SET
if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM)
   && IS_ATTRIBUTE(attributes, TPMA_OBJECT, encryptedDuplication))
   return TPM_RCS_ATTRIBUTES;

// If a parent object has fixedTPM CLEAR, the child must have the
// same encryptedDuplication value as its parent.
// Primary objects are considered to have a fixedTPM parent (the seeds).
if(parentObject != NULL
   && !IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, fixedTPM))
   {
      if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, encryptedDuplication)
         != IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, encryptedDuplication))
         return TPM_RCS_ATTRIBUTES;
   }

// Special checks for derived objects
if((parentObject != NULL) && (parentObject->attributes.derivation == SET))
   {
      // A derived object has the same settings for fixedTPM as its parent
      if(IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedTPM)
         != IS_ATTRIBUTE(parentAttributes, TPMA_OBJECT, fixedTPM))
         return TPM_RCS_ATTRIBUTES;
      // A derived object is required to be fixedParent
      if(!IS_ATTRIBUTE(attributes, TPMA_OBJECT, fixedParent))
         return TPM_RCS_ATTRIBUTES;
   }
   return SchemeChecks(parentObject, publicArea);
}

7.6.3.6 FillInCreationData()

Fill in creation data for an object.

void FillInCreationData(
   TPMI_DH_OBJECT parentHandle, // IN: handle of parent
   TPMI_ALG_HASH nameHashAlg,  // IN: name hash algorithm
   TPML_PCR_SELECTION *creationPCR, // IN: PCR selection
   TPM2B_DATA *outsideData,  // IN: outside data
   TPM2B_CREATION_DATA *outCreation, // OUT: creation data for output
   TPM2B_DIGEST *creationDigest // OUT: creation digest
   )
{
   BYTE creationBuffer[sizeof(TPMS_CREATION_DATA)];
   BYTE *buffer;
   HASH_STATE hashState;

   // Fill in TPMS_CREATION_DATA in outCreation
   // Compute PCR digest
   PCRComputeCurrentDigest(nameHashAlg, creationPCR,
                           &outCreation->creationData.pcrDigest);
   // Put back PCR selection list
   outCreation->creationData.pcrSelect = *creationPCR;
   // Get locality
   outCreation->creationData.locality
      = LocalityGetAttributes(_plat__LocalityGet());
   outCreation->creationData.parentNameAlg = TPM_ALG_NULL;
556  // If the parent is either a primary seed or TPM_ALG_NULL, then the Name
557  // and QN of the parent are the parent's handle.
558  if(HandleGetType(parentHandle) == TPM_HT_PERMANENT)
559  {
560      buffer = &outCreation->creationData.parentName.t.name[0];
561      outCreation->creationData.parentName.t.size =
562          TPM_HANDLE_Marsh(&parentHandle, &buffer, NULL);
563      // For a primary or temporary object, the parent name (a handle) and the
564      // parent's QN are the same
565      outCreation->creationData.parentQualifiedName
566          = outCreation->creationData.parentName;
567  }
568  else // Regular object
569  {
570      OBJECT          *parentObject = HandleToObject(parentHandle);
571      // Set name algorithm
572      outCreation->creationData.parentNameAlg =
573          parentObject->publicArea.nameAlg;
574      // Copy parent name
575      outCreation->creationData.parentName = parentObject->name;
576      // Copy parent qualified name
577      outCreation->creationData.parentQualifiedName =
578          parentObject->qualifiedName;
579  }
580  // Copy outside information
581  outCreation->creationData.outsideInfo = *outsideData;
582  // Marshal creation data to canonical form
583  buffer = creationBuffer;
584  outCreation->size = TPMS_CREATION_DATA_Marsh(&outCreation->creationData,
585      &buffer, NULL);
586  // Compute hash for creation field in public template
587  creationDigest->t.size = CryptHashStart(&hashState, nameHashAlg);
588  CryptDigestUpdate(&hashState, outCreation->size, creationBuffer);
589  CryptHashEnd2B(&hashState, &creationDigest->b);
590  return;
591 }
592
7.6.3.7 GetSeedForKDF()

Get a seed for KDF. The KDF for encryption and HMAC key use the same seed.

597  const TPM2B *  
598  GetSeedForKDF("  
599      OBJECT          *protector     // IN: the protector handle
600      )
601  {
602      // Get seed for encryption key. Use input seed if provided.
603      // Otherwise, using protector object's seedValue. TPM_RH_NULL is the only
604      // exception that we may not have a loaded object as protector. In such a
605      // case, use nullProof as seed.
606      if(protector == NULL)
607          return &gr.nullProof.b;
608      else
609          return &protector->sensitive.seedValue.b;
610  }
7.6.3.8 ProduceOuterWrap()

This function produce outer wrap for a buffer containing the sensitive data. It requires the sensitive data being marshaled to the `outerBuffer`, with the leading bytes reserved for integrity hash. If iv is used, iv space should be reserved at the beginning of the buffer. It assumes the sensitive data starts at address `(outerBuffer + integrity size (+ iv size))`. This function performs:

a) Add IV before sensitive area if required  
b) encrypt sensitive data, if iv is required, encrypt by iv. otherwise, encrypted by a NULL iv  
c) add HMAC integrity at the beginning of the buffer It returns the total size of blob with outer wrap

```c
UINT16 ProduceOuterWrap(
  OBJECT *protector,  // IN: The handle of the object that provides
  // protection. For object, it is parent
  // handle. For credential, it is the handle
  // of encrypt object.
  TPM2B *name,  // IN: the name of the object
  TPM_ALG_ID hashAlg,  // IN: hash algorithm for outer wrap
  TPM2B *seed,  // IN: an external seed may be provided for
  // duplication blob. For non duplication
  // blob, this parameter should be NULL
  BOOL useIV,  // IN: indicate if an IV is used
  UINT16 dataSize,  // IN: the size of sensitive data, excluding the
  // leading integrity buffer size or the
  // optional iv size
  BYTE *outerBuffer  // IN/OUT: outer buffer with sensitive data in
  // it )
{
  TPM_ALG_ID symAlg;
  UINT16 keyBits;
  TPM2B_SYM_KEY symKey;
  TPM2B_IV ivRNG;  // IV from RNG
  TPM2B_IV *iv = NULL;
  UINT16 ivSize = 0;  // size of iv area, including the size field
  BYTE *sensitiveData;  // pointer to the sensitive data
  TPM2B_DIGEST integrity;
  UINT16 integritySize;
  BYTE *buffer;  // Auxiliary buffer pointer

  // Compute the beginning of sensitive data. The outer integrity should
  // always exist if this function is called to make an outer wrap
  integritySize = sizeof(UINT16) + CryptHashGetDigestSize(hashAlg);
  sensitiveData = outerBuffer + integritySize;

  if(useIV) {
    ivSize = GetIV2BSize(protector);

    // Generate IV from RNG. The iv data size should be the total IV area
    // size minus the size of size field
    ivRNG.t.size = ivSize - sizeof(UINT16);
    CryptRandomGenerate(ivRNG.t.size, ivRNG.t.buffer);

    // Marshal IV to buffer
    buffer = sensitiveData;
    TPM2B_IV.Marshal(&ivRNG, &buffer, NULL);

    // adjust sensitive data starting after IV area
    sensitiveData += ivSize;
  }
```
7.6.3.9 UnwrapOuter()

This function remove the outer wrap of a blob containing sensitive data. This function performs:

a) check integrity of outer blob

b) decrypt outer blob

Error Returns | Meaning
---|---
TPM_RCS_INSUFFICIENT | error during sensitive data unmarshaling
TPM_RCS_INTEGRITY | sensitive data integrity is broken
TPM_RCS_SIZE | error during sensitive data unmarshaling
TPM_RCS_VALUE | IV size for CFB does not match the encryption algorithm block size

```
TPM_RC UnwrapOuter(
    OBJECT          *protector, // IN: The object that provides
    TPM2B           *name,     // protection. For object, it is parent
    TPM_ALG_ID      hashAlg,   // handle. For credential, it is the
    TPM2B           *seed,     // encrypt object.
    BOOL             useIV,     // IN: the name of the object
    TPM2B           *name,     // IN: hash algorithm for outer wrap
    TPM2B           *seed,     // IN: an external seed may be provided for
    TPM2B           *seed,     // duplication blob. For non duplication
    TPM2B           *seed,     // blob, this parameter should be NULL.
    TPM2B           *seed,     // IN: indicates if an IV is used
    UINT16           dataSize, // IN: size of sensitive data in outerBuffer,
    UINT16           dataSize, // including the leading integrity buffer
    BYTE            *outerBuffer // IN/OUT: sensitive data
)
```
// Unmarshal integrity
sensitiveData = outerBuffer;
size = (INT32)dataSize;
result = TPM2B_DIGEST_Unmarshal(&integrity, &sensitiveData, &size);
if (result == TPM_RC_SUCCESS) {
    // Compute integrity to compare
    ComputeOuterIntegrity(name, protector, hashAlg, seed,
    (UINT16)size, sensitiveData,
    &integrityToCompare);
    // Compare outer blob integrity
    if (!MemoryEqual2B(&integrity.b, &integrityToCompare.b))
        return TPM_RC_INTEGRITY;
    // Get the symmetric algorithm parameters used for encryption
    ComputeProtectionKeyParms(protector, hashAlg, name, seed,
    &symAlg, &keyBits, &symKey);
    // Retrieve IV if it is used
    if (useIV) {
        result = TPM2B_IV_Unmarshal(&ivIn, &sensitiveData, &size);
        if (result == TPM_RC_SUCCESS) {
            // The input iv size for CFB must match the encryption algorithm
            // block size
            if (ivIn.t.size != CryptGetSymmetricBlockSize(symAlg, keyBits))
                result = TPM_RC_VALUE;
            else
                iv = &ivIn;
        }
    }
    // If no errors, decrypt private in place. Since this function uses CFB,
    // CryptSymmetricDecrypt() will not return any errors. It may fail but it will
    // not return an error.
    if (result == TPM_RC_SUCCESS)
        CryptSymmetricDecrypt(sensitiveData, symAlg, keyBits,
        symKey.t.buffer, iv, TPM_ALG_CFB,
        (UINT16)size, sensitiveData);
    return result;
}

7.6.3.10 MarshalSensitive()

This function is used to marshal a sensitive area. Among other things, it adjusts the size of the authValue
to be no smaller than the digest of nameAlg Returns the size of the marshaled area.

static UINT16
MarshalSensitive(
    BYTE                *buffer,  // OUT: receiving buffer
    TPMT_SENSITIVE      *sensitive,  // IN: the sensitive area to marshal
    TPMI_ALG_HASH      nameAlg     // IN:
) {
    BYTE                *sizeField = buffer;  // saved so that size can be
    UINT16               retVal;
    // marshaled after it is known
    // Pad the authValue if needed
    MemoryPad2B(&sensitive->authValue.b, CryptHashGetDigestSize(nameAlg));
    buffer += 2;
7.6.3.11 SensitiveToPrivate()

This function prepare the private blob for off the chip storage. The operations in this function:

a) marshal TPM2B_SENSITIVE structure into the buffer of TPM2B_PRIVATE

b) apply encryption to the sensitive area.

c) apply outer integrity computation.

```c
void SensitiveToPrivate(
    TPMT_SENSITIVE *sensitive,   // IN: sensitive structure
    TPM2B *name,                // IN: the name of the object
    OBJECT *parent,             // IN: The parent object
    TPM_ALG_ID nameAlg,         // IN: hash algorithm in public area. This
                                // parameter is used when parentHandle is
                                // NULL, in which case the object is
                                // temporary.
    TPM2B_PRIVATE *outPrivate   // OUT: output private structure
)
{
    BYTE *sensitiveData;        // pointer to the sensitive data
    UINT16 dataSize;            // data blob size
    TPMI_ALG_HASH hashAlg;      // hash algorithm for integrity
    UINT16 integritySize;       // hash algorithm for integrity
    UINT16 ivSize;
    // pAssert(name != NULL && name->size != 0);
    // Find the hash algorithm for integrity computation
    if(name != NULL)
    {
        // For Temporary Object, using self name algorithm
        hashAlg = nameAlg;
    }
    else
    {
        // Otherwise, using parent's name algorithm
        hashAlg = ObjectGetNameAlg(parent);
    }
    // Starting of sensitive data without wrappers
    sensitiveData = outPrivate->t.buffer;
    // Compute the integrity size
    integritySize = sizeof(UINT16) + CryptHashGetDigestSize(hashAlg);
    // Reserve space for integrity
    sensitiveData += integritySize;
    // Get iv size
    ivSize = GetIV2BSIZE(parent);
    // Reserve space for iv
    sensitiveData += ivSize;
```
821  // Marshal the sensitive area including authValue size adjustments.
822  dataSize = MarshalSensitive(sensitiveData, sensitive, nameAlg);
823
824  //Produce outer wrap, including encryption and HMAC
825  outPrivate->t.size = ProduceOuterWrap(parent, name, hashAlg, NULL,
826                                TRUE, dataSize, outPrivate->t.buffer);
827  return;
828 }
829

7.6.3.12 PrivateToSensitive()

Unwrap a input private area. Check the integrity, decrypt and retrieve data to a sensitive structure. The operations in this function:

a) check the integrity HMAC of the input private area
b) decrypt the private buffer
c) unmarshal TPMT_SENSITIVE structure into the buffer of TPMT_SENSITIVE

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RCS_INTEGRITY</td>
<td>if the private area integrity is bad</td>
</tr>
<tr>
<td>TPM_RC_SENSITIVE</td>
<td>unmarshal errors while unmarshaling TPMS_ENCRYPT from input private</td>
</tr>
<tr>
<td>TPM_RCS_SIZE</td>
<td>error during sensitive data unmarshaling</td>
</tr>
<tr>
<td>TPM_RCS_VALUE</td>
<td>outer wrapper does not have an iV of the correct size</td>
</tr>
</tbody>
</table>

TPM_RC

830  PrivateToSensitive(
831  TPM2B *inPrivate,       // IN: input private structure
832  TPM2B *name,            // IN: the name of the object
833  OBJECT *parent,         // IN: parent object
834  TPM_ALG_ID nameAlg,     // IN: hash algorithm in public area. It is
835  // passed separately because we only pass
836  // name, rather than the whole public area
837  // of the object. This parameter is used in
838  // the following two cases: 1. primary
839  // objects. 2. duplication blob with inner
840  // wrap. In other cases, this parameter
841  // will be ignored
842  TPM_TPM_SENSITIVE *sensitive   // OUT: sensitive structure
843 )
844 {
845    TPM_RC result;          // OUT: sensitive structure
846    BYTE *buffer;
847    INT32 size;
848    BYTE *sensitiveData;   // pointer to the sensitive data
849    UINT16 dataSize;
850    UINT16 dataSizeInput;
851    TPM_ALG_HASH hashAlg;   // hash algorithm for integrity
852    UINT16 integritySize;
853    UINT16 ivSize;
854
855    // Make sure that name is provided
856    pAssert(name != NULL && name->size != 0);
857
858    // Find the hash algorithm for integrity computation
859    if(parent == NULL)
860    {
861      // For Temporary Object, using self name algorithm
862      hashAlg = nameAlg;
863      // For Temporary Object, using self name algorithm
864    }
else
{
    // Otherwise, using parent's name algorithm
    hashAlg = ObjectGetNameAlg(parent);
}

// unwrap outer
result = UnwrapOuter(parent, name, hashAlg, NULL, TRUE,
inPrivate->size, inPrivate->buffer);
if(result != TPM_RC_SUCCESS)
    return result;

// Compute the inner integrity size.
integritySize = sizeof(UINT16) + CryptHashGetDigestSize(hashAlg);

// Get iv size
ivSize = GetIV2BSIZE(parent);

// The starting of sensitive data and data size without outer wrapper
sensitiveData = inPrivate->buffer + integritySize + ivSize;
dataSize = inPrivate->size - integritySize - ivSize;

// Unmarshal input data size
buffer = sensitiveData;
size = (INT32)dataSize;
result = UINT16_Unmarshal(&dataSizeInput, &buffer, &size);
if(result == TPM_RC_SUCCESS)
{
    if((dataSizeInput + sizeof(UINT16)) != dataSize)
        result = TPM_RC_SENSITIVE;
    else
    {
        // Unmarshal sensitive buffer to sensitive structure
        result = TPMT_SENSITIVE_Unmarshal(sensitive, &buffer, &size);
        if(result != TPM_RC_SUCCESS || size != 0)
        {
            result = TPM_RC_SENSITIVE;
        }
    }
}
return result;

7.6.3.13 SensitiveToDuplicate()

This function prepare the duplication blob from the sensitive area. The operations in this function:
a) marshal TPMT_SENSITIVE structure into the buffer of TPM2B_PRIVATE
b) apply inner wrap to the sensitive area if required
c) apply outer wrap if required

void SensitiveToDuplicate(
    TPMT_SENSITIVE *sensitive,  // IN: sensitive structure
    TPM2B *name,               // IN: the name of the object
    OBJECT *parent,            // IN: The new parent object
    TPM_ALG_ID nameAlg,       // IN: hash algorithm in public area. It is passed separately because we only pass name, rather than the whole public area of the object.
    TPM2B *seed,              // IN: the external seed. If external seed is provided with size of 0, no outer wrap should be applied
to duplication blob.
TPMT_SYM_DEF_OBJECT *symDef,  // IN: Symmetric key definition. If the
   // symmetric key algorithm is NULL,
   // no inner wrap should be applied.
TPM2B_DATA *innerSymKey,  // IN/OUT: a symmetric key may be
   // provided to encrypt the inner
   // wrap of a duplication blob. May
   // be generated here if needed.
TPM2B_PRIVATE *outPrivate  // OUT: output private structure
{

BYTE *sensitiveData;  // pointer to the sensitive data

TPMI_ALG_HASH outerHash = TPM_ALG_NULL;  // The hash algorithm for outer wrap
TPMI_ALG_HASH innerHash = TPM_ALG_NULL;  // The hash algorithm for inner wrap
UINT16 dataSize;  // data blob size

BOOL doInnerWrap = FALSE;
BOOL doOuterWrap = FALSE;

// Make sure that name is provided
pAssert(name != NULL && name->size != 0);

// Make sure symDef and innerSymKey are not NULL
pAssert(symDef != NULL && innerSymKey != NULL);

// Starting of sensitive data without wrappers
sensitiveData = outPrivate->t.buffer;

// Find out if inner wrap is required
if(symDef->algorithm != TPM_ALG_NULL)
{
doInnerWrap = TRUE;

 // Use self nameAlg as inner hash algorithm
innerHash = nameAlg;

 // Adjust sensitive data pointer
sensitiveData += sizeof(UINT16) + CryptHashGetDigestSize(innerHash);
}

// Find out if outer wrap is required
if(seed->size != 0)
{
doOuterWrap = TRUE;

 // Use parent nameAlg as outer hash algorithm
outerHash = ObjectGetNameAlg(parent);

 // Adjust sensitive data pointer
sensitiveData += sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
}

// Marshal sensitive area
dataSize = MarshalSensitive(sensitiveData, sensitive, nameAlg);

// Apply inner wrap for duplication blob. It includes both integrity and
// encryption
if(doInnerWrap)
{
BYTE *innerBuffer = NULL;
BOOL symKeyInput = TRUE;
innerBuffer = outPrivate->t.buffer;

// Skip outer integrity space
if(doOuterWrap)
   innerBuffer += sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
dataSize = ProduceInnerIntegrity(name, innerHash, dataSize,
   innerBuffer);

// Generate inner encryption key if needed
if(innerSymKey->t.size == 0)
   {

innerSymKey->t.size = (symDef->keyBits.sym + 7) / 8;
CryptRandomGenerate(innerSymKey->t.size, innerSymKey->t.buffer);

// TPM generates symmetric encryption. Set the flag to FALSE
symKeyInput = FALSE;
}
else
{
    // assume the input key size should matches the symmetric definition
    pAssert(innerSymKey->t.size == (symDef->keyBits.sym + 7) / 8);
}

// Encrypt inner buffer in place
CryptSymmetricEncrypt(innerBuffer, symDef->algorithm,
symDef->keyBits.sym, innerSymKey->t.buffer, NULL,
TPM_ALG_CFB, dataSize, innerBuffer);

// If the symmetric encryption key is imported, clear the buffer for
// output
if(symKeyInput)
innerSymKey->t.size = 0;

// Apply outer wrap for duplication blob. It includes both integrity and
// encryption
if(doOuterWrap)
{
    dataSize = ProduceOuterWrap(parent, name, outerHash, seed, FALSE,
dataSize, outPrivate->t.buffer);
}

// Data size for output
outPrivate->t.size = dataSize;
return;

7.6.3.14 DuplicateToSensitive()

Unwrap a duplication blob. Check the integrity, decrypt and retrieve data to a sensitive structure. The
operations in this function:
a) check the integrity HMAC of the input private area
b) decrypt the private buffer
c) unmarshal TPMT_SENSITIVE structure into the buffer of TPMT_SENSITIVE

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_INSUFFICIENT</td>
<td>unmarshaling sensitive data from inPrivate failed</td>
</tr>
<tr>
<td>TPM_RC_INTEGRITY</td>
<td>inPrivate data integrity is broken</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>unmarshaling sensitive data from inPrivate failed</td>
</tr>
</tbody>
</table>

TPM_RC

duplicateToSensitive(

    *inPrivate,       // IN: input private structure
    *name,            // IN: the name of the object
    *parent,          // IN: the parent
    nameAlg,          // IN: hash algorithm in public area.
    *seed,            // IN: an external seed may be provided.
    // If external seed is provided with
    // size of 0, no outer wrap is
    applied
    symDef,          // IN: Symmetric key definition. If the
)
typedef TPM2B   *innerSymKey;  // IN: a symmetric key may be provided
to decrypt the inner wrap of a duplication blob.

typedef TPMT_SENSITIVE   *sensitive;  // OUT: sensitive structure

/*
   // symmetric key algorithm is NULL,
   // no inner wrap is applied
   // Make sure name is provided
   pAssert(name != NULL && name->size != 0);
   // Make sure symDef and innerSymKey are not NULL
   pAssert(symDef != NULL && innerSymKey != NULL);
   // Starting of sensitive data
   sensitiveData = inPrivate->buffer;
dataSize = inPrivate->size;
   // Find out if outer wrap is applied
   if(seed->size != 0)
   {
      // Use parent nameAlg as outer hash algorithm
      TPMI_ALG_HASH   outerHash = parent->publicArea.nameAlg;
      result = UnwrapOuter(parent, name, outerHash, seed, FALSE,
dataSize, sensitiveData);
      if(result != TPM_RC_SUCCESS)
      return result;
      // Adjust sensitive data pointer and size
      sensitiveData += sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
dataSize -= sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
   }
   // Find out if inner wrap is applied
   if(symDef->algorithm != TPM_ALG_NULL)
   {
      // assume the input key size matches the symmetric definition
      pAssert(innerSymKey->size == (symDef->keyBits.sym + 7) / 8);
      // Decrypt inner buffer in place
      CryptSymmetricDecrypt(sensitiveData, symDef->algorithm,
symDef->keyBits.sym, innerSymKey->buffer, NULL,
TPM_ALG_CFB, dataSize, sensitiveData);
      // Check inner integrity
      result = CheckInnerIntegrity(name, nameAlg, dataSize, sensitiveData);
      if(result != TPM_RC_SUCCESS)
      return result;
      // Adjust sensitive data pointer and size
      sensitiveData += sizeof(UINT16) + CryptHashGetDigestSize(nameAlg);
dataSize -= sizeof(UINT16) + CryptHashGetDigestSize(nameAlg);
   }
   // Unmarshal input data size
   buffer = sensitiveData;
   size = (INT32)dataSize;
   result = UINT16_Unmarshal(&dataSizeInput, &buffer, &size);
   if(result == TPM_RC_SUCCESS)
   {
      if((dataSizeInput + sizeof(UINT16)) != dataSize)
      result = TPM_RC_SIZE;
   }
*/
7.6.3.15 SecretToCredential()

This function prepare the credential blob from a secret (a TPM2B_DIGEST) The operations in this function:

a) marshal TPM2B_DIGEST structure into the buffer of TPM2B_ID_OBJECT
b) encrypt the private buffer, excluding the leading integrity HMAC area
c) compute integrity HMAC and append to the beginning of the buffer.
d) Set the total size of TPM2B_ID_OBJECT buffer

```c
void SecretToCredential(TPM2B_DIGEST *secret, // IN: secret information
                      TPM2B *name,    // IN: the name of the object
                      TPM2B *seed,    // IN: an external seed.
                      OBJECT *protector, // IN: the protector
                      TPM2B_ID_OBJECT *outIDObject // OUT: output credential
                      )
{
    BYTE *buffer;  // Auxiliary buffer pointer
    BYTE *sensitiveData; // pointer to the sensitive data
    TPMI_ALG_HASH outerHash; // The hash algorithm for outer wrap
    UINT16 dataSize; // data blob size

    // pAssert(secret != NULL && outIDObject != NULL);
    // use protector's name algorithm as outer hash
    outerHash = ObjectGetNameAlg(protector);
    // Marshal secret area to credential buffer, leave space for integrity
    sensitiveData = outIDObject->t.credential + sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
    // Marshal secret area
    buffer = sensitiveData;
    dataSize = TPM2B_DIGEST_Marshal(secret, &buffer, NULL);
    // Apply outer wrap
    outIDObject->t.size = ProduceOuterWrap(protector, name, outerHash, seed, FALSE,
                        dataSize, outIDObject->t.credential);
    return;
}
```

7.6.3.16 CredentialToSecret()

Unwrap a credential. Check the integrity, decrypt and retrieve data to a TPM2B_DIGEST structure. The operations in this function:
a) check the integrity HMAC of the input credential area
b) decrypt the credential buffer
c) unmarshal TPM2B_DIGEST structure into the buffer of TPM2B_DIGEST

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_INSUFFICIENT</td>
<td>error during credential unmarshaling</td>
</tr>
<tr>
<td>TPM_RC_INTEGRITY</td>
<td>credential integrity is broken</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>error during credential unmarshaling</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>IV size does not match the encryption algorithm block size</td>
</tr>
</tbody>
</table>

```
1137  TPM_RC
1138  CredentialToSecret(
1139      TPM2B *inIDObject, // IN: input credential blob
1140      TPM2B *name,     // IN: the name of the object
1141      TPM2B *seed,     // IN: an external seed.
1142      OBJECT *protector, // IN: the protector
1143      TPM2B_DIGEST *secret // OUT: secret information
1144  )
1145  {  
1146      TPM_RC   result;
1147      BYTE     *buffer;
1148      INT32    size;
1149      TPMI_ALG_HASH outerHash; // The hash algorithm for outer wrap
1150      BYTE     *sensitiveData; // pointer to the sensitive data
1151      UINT16   dataSize;
1152      // use protector's name algorithm as outer hash
1153      outerHash = ObjectGetNameAlg(protector);
1154      // Unwrap outer, a TPM_RC_INTEGRITY error may be returned at this point
1155      result = UnwrapOuter(protector, name, outerHash, seed, FALSE,
1156                              inIDObject->size, inIDObject->buffer);
1157      if(result == TPM_RC_SUCCESS)
1158      {
1159          // Compute the beginning of sensitive data
1160          sensitiveData = inIDObject->buffer
1161          + sizeof(UINT16) + CryptHashGetDigestSize(outerHash);
1162          dataSize = inIDObject->size
1163          - (sizeof(UINT16) + CryptHashGetDigestSize(outerHash));
1164          // Unmarshal secret buffer to TPM2B_DIGEST structure
1165          buffer = sensitiveData;
1166          size = (INT32)dataSize;
1167          result = TPM2B_DIGEST_Unmarshal(secret, &buffer, &size);
1168          // If there were no other unmarshaling errors, make sure that the
1169          // expected amount of data was recovered
1170          if(result == TPM_RC_SUCCESS && size != 0)
1171              return TPM_RC_SIZE;
1172      }
1173      return result;
1174  }
```

**7.6.3.17 MemoryRemoveTrailingZeros()**

This function is used to adjust the length of an authorization value. It adjusts the size of the TPM2B so that it does not include octets at the end of the buffer that contain zero. The function returns the number of non-zero octets in the buffer.

```
1178  UINT16
```
MemoryRemoveTrailingZeros(
    TPM2B_AUTH *auth,  // IN/OUT: value to adjust
)
{
    while((auth->t.size > 0) && (auth->t.buffer[auth->t.size - 1] == 0))
        auth->t.size--;
    return auth->t.size;
}

7.6.3.18 SetLabelAndContext()

This function sets the label and context for a derived key. It is possible that label or context can end up being an Empty Buffer.

TPM_RC
SetLabelAndContext(
    TPMS_DERIVE         *labelContext,  // IN/OUT: the recovered label and context
    TPM2B_SENSITIVE_DATA *sensitive     // IN: the sensitive data
)
{
    TPMS_DERIVE sensitiveValue;
    TPM_RC result;
    INT32 size;
    BYTE *buff;
    // Unmarshal a TPMS_DERIVE from the TPM2B_SENSITIVE_DATA buffer
    // If there is something to unmarshal...
    if(sensitive->t.size != 0)
    {
        size = sensitive->t.size;
        buff = sensitive->t.buffer;
        result = TPMS_DERIVE_Unmarshal(&sensitiveValue, &buff, &size);
        if(result != TPM_RC_SUCCESS)
            return result;
        // If there was a label in the public area leave it there, otherwise, copy the new value
        if(labelContext->label.t.size == 0)
            MemoryCopy2B(&labelContext->label.b, &sensitiveValue.label.b, sizeof(labelContext->label.t.buffer));
        // if there was a context string in publicArea, it overrides
        if(labelContext->context.t.size == 0)
            MemoryCopy2B(&labelContext->context.b, &sensitiveValue.context.b, sizeof(labelContext->context.t.buffer));
        return TPM_RC_SUCCESS;
    }
}

7.6.3.19 UnmarshalToPublic()

Support function to unmarshal the template. This is used because the Input may be a TPMT_TEMPLATE and that structure does not have the same size as a TPMT_PUBLIC because of the difference between the unique and seed fields. If derive is not NULL, then the seed field is assumed to contain a label and context that are unmarshaled into derive.

TPM_RC
UnmarshalToPublic(
    TPMT_PUBLIC         *tOut,    // OUT: output
    TPMT2B_TEMPLATE     *tIn,     // IN:
    BOOL derivation,    // IN: indicates if this is for a derivation
    TPMS_DERIVE         *labelContext// OUT: label and context if derivation
)
{  
  BYTE  *buffer = tIn->t.buffer;  
  INT32  size = tIn->t.size;  
  TPM_RC  result;  
  
  // make sure that tOut is zeroed so that there are no remnants from previous  
  // uses  
  MemorySet(tOut, 0, sizeof(TPMT_PUBLIC));  
  // Unmarshal the components of the TPMT_PUBLIC up to the unique field  
  result = TPMI_ALG_PUBLIC_Unmarshal(&tOut->type, &buffer, &size);  
  if(result != TPM_RC_SUCCESS)  
    return result;  
  result = TPMI_ALG_HASH_Unmarshal(&tOut->nameAlg, &buffer, &size, FALSE);  
  if(result != TPM_RC_SUCCESS)  
    return result;  
  result = TPMA_OBJECT_Unmarshal(&tOut->objectAttributes, &buffer, &size);  
  if(result != TPM_RC_SUCCESS)  
    return result;  
  result = TPM2B_DIGEST_Unmarshal(&tOut->authPolicy, &buffer, &size);  
  if(result != TPM_RC_SUCCESS)  
    return result;  
  result = TPMU_PUBLIC_PARMS_Unmarshal(&tOut->parameters, &buffer, &size,  
                                         tOut->type);  
  if(result != TPM_RC_SUCCESS)  
    return result;  
  // Now unmarshal a TPMS_DERIVE if this is for derivation  
  if(derivation)  
    result = TPMS_DERIVE_Unmarshal(labelContext, &buffer, &size);  
  else  
    // otherwise, unmarshal a TPMU_PUBLIC_ID  
    result = TPMU_PUBLIC_ID_Unmarshal(&tOut->unique, &buffer, &size,  
                                       tOut->type);  
  // Make sure the template was used up  
  if((result == TPM_RC_SUCCESS) && (size != 0))  
    result = TPM_RC_SIZE;  
  return result;  
}

7.6.3.20 ObjectSetExternal()

Set the external attributes for an object.

void ObjectSetExternal(
  OBJECT *object
)
{
  object->attributes.external = SET;
}
7.7 Encrypt Decrypt Support (EncryptDecrypt_spt.c)

```c
#include "Tpm.h"
#include "EncryptDecrypt_fp.h"
#include "EncryptDecrypt_spt_fp.h"
#if CC_EncryptDecrypt2

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>is not a symmetric decryption key with both public and private portions loaded</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>IvIn size is incompatible with the block cipher mode; or inData size is not an even multiple of the block size for CBC or ECB mode</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>keyHandle is restricted and the argument mode does not match the key's mode</td>
</tr>
</tbody>
</table>

TPM_RC

EncryptDecryptShared(
    TPMI_DH_OBJECT keyHandleIn,
    TPMI_YES_NO decryptIn,
    TPMI_ALG_SYM_MODE modeIn,
    TPM2B_IV *ivIn,
    TPM2B_MAX_BUFFER *inData,
    EncryptDecrypt_Out *out
) {
    OBJECT *symKey;
    UINT16 keySize;
    UINT16 blockSize;
    BYTE *key;
    TPM_ALG_ID alg;
    TPM_ALG_ID mode;
    TPM_RC result;
    BOOL OK;

    // Input Validation
    symKey = HandleToObject(keyHandleIn);
    mode = symKey->publicArea.parameters.symDetail.sym.mode.sym;

    // The input key should be a symmetric key
    if Tümkey->publicArea.type != TPM_ALG_SYMCIPHER)
        return TPM_RCS_KEY + RC_EncryptDecrypt_keyHandle;

    // The key must be unrestricted and allow the selected operation
    OK = IS_ATTRIBUTE(tumkey->publicArea.objectAttributes,
                      TPMA_OBJECT, restricted);
    if YES == decryptIn)
        OK = OK && IS_ATTRIBUTE(tumkey->publicArea.objectAttributes,
                      TPMA_OBJECT, decrypt);
    else
        OK = OK && IS_ATTRIBUTE(tumkey->publicArea.objectAttributes,
                      TPMA_OBJECT, sign);

    if(!OK)
        return TPM_RCS_ATTRIBUTES + RC_EncryptDecrypt_keyHandle;

    // Make sure that key is an encrypt/decrypt key and not SMAC
    if(!CryptSymModeIsValid(mode, TRUE))
        return TPM_RCS_MODE + RC_EncryptDecrypt_keyHandle;

    // If the key mode is not TPM_ALG_NULL...
    if(mode != TPM_ALG_NULL)
    {
        // then the input mode has to be TPM_ALG_NULL or the same as the key
```

```c
```
```c
if ((modeIn != TPM_ALG_NULL) && (modeIn != mode))
    return TPM_RCS_MODE + RC_EncryptDecrypt_mode;
else
    // if the key mode is null, then the input can't be null
    if (modeIn == TPM_ALG_NULL)
        return TPM_RCS_MODE + RC_EncryptDecrypt_mode;
    mode = modeIn;
}
// The input iv for ECB mode should be an Empty Buffer. All the other modes
// should have an iv size same as encryption block size
keySize = symKey->publicArea.parameters.symDetail.sym.keyBits.sym;
alg = symKey->publicArea.parameters.symDetail.sym.algorithm;
blockSize = CryptGetSymmetricBlockSize(alg, keySize);

// reverify the algorithm. This is mainly to keep static analysis tools happy
if (blockSize == 0)
    return TPM_RCS_KEY + RC_EncryptDecrypt_keyHandle;
// Note: When an algorithm is not supported by a TPM, the TPM_ALG_xxx for that
// algorithm is not defined. However, it is assumed that the ALG_xxx(VALUE for
// the algorithm is always defined. Both have the same numeric value.
// ALG_xxx(VALUE is used here so that the code does not get cluttered with
// #ifdef's. Having this check does not mean that the algorithm is supported.
// If it was not supported the unmarshaling code would have rejected it before
// this function were called. This means that, depending on the implementation,
// the check could be redundant but it doesn't hurt.
if (((mode == ALG_ECB_VALUE) && (ivIn->t.size != 0))
    || ((mode != ALG_ECB_VALUE) && (ivIn->t.size != blockSize))
    return TPM_RCS_SIZE + RC_EncryptDecrypt_ivIn;
// The input data size of CBC mode or ECB mode must be an even multiple of
// the symmetric algorithm's block size
if (((mode == ALG_CBC_VALUE) || (mode == ALG_ECB_VALUE))
    && ((inData->t.size % blockSize) != 0))
    return TPM_RCS_SIZE + RC_EncryptDecrypt_inData;
// Copy IV
// Note: This is copied here so that the calls to the encrypt/decrypt functions
// will modify the output buffer, not the input buffer
out->ivOut = *ivIn;

// Command Output
key = symKey->sensitive.sensitive.sym.t.buffer;
// For symmetric encryption, the cipher data size is the same as plain data
// size.
out->outData.t.size = inData->t.size;
if (decryptIn == YES)
{
    // Decrypt data to output
    result = CryptSymmetricDecrypt(out->outData.t.buffer, alg, keySize, key,
        &(out->ivOut), mode, inData->t.size,
        inData->t.buffer);
}
else
{
    // Encrypt data to output
    result = CryptSymmetricEncrypt(out->outData.t.buffer, alg, keySize, key,
        &(out->ivOut), mode, inData->t.size,
        inData->t.buffer);
}
return result;
#endif // CC_EncryptDecrypt
```
8 Subsystem

8.1 CommandAudit.c

8.1.1 Introduction

This file contains the functions that support command audit.

8.1.2 Includes

```c
#include "Tpm.h"
```

8.1.3 Functions

8.1.3.1 CommandAuditPreInstall_Init()

This function initializes the command audit list. This function is simulates the behavior of manufacturing. A function is used instead of a structure definition because this is easier than figuring out the initialization value for a bit array.

This function would not be implemented outside of a manufacturing or simulation environment.

```c
void CommandAuditPreInstall_Init()
{
    // Clear all the audit commands
    MemorySet(gp.auditCommands, 0x00, sizeof(gp.auditCommands));

    // TPM CC_SetCommandCodeAuditStatus always being audited
    CommandAuditSet(TPM_CC_SetCommandCodeAuditStatus);

    // Set initial command audit hash algorithm to be context integrity hash
    // algorithm
    gp.auditHashAlg = CONTEXT_INTEGRITY_HASH_ALG;

    // Set up audit counter to be 0
    gp.auditCounter = 0;

    // Write command audit persistent data to NV
    NV_SYNC_PERSISTENT(auditCommands);
    NV_SYNC_PERSISTENT(auditHashAlg);
    NV_SYNC_PERSISTENT(auditCounter);

    return;
}
```

8.1.3.2 CommandAuditStartup()

This function clears the command audit digest on a TPM Reset.

```c
void CommandAuditStartup(
    STARTUP_TYPE type       // IN: start up type
)
{
    if((type != SU_RESTART) && (type != SU_RESUME))
```
8.1.3.3 CommandAuditSet()

This function will SET the audit flag for a command. This function will not SET the audit flag for a command that is not implemented. This ensures that the audit status is not SET when TPM2_GetCapability() is used to read the list of audited commands.

This function is only used by TPM2_SetCommandCodeAuditStatus().

The actions in TPM2_SetCommandCodeAuditStatus() are expected to cause the changes to be saved to NV after it is setting and clearing bits.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>command code audit status was changed</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>command code audit status was not changed</td>
</tr>
</tbody>
</table>

```c
BOOL CommandAuditSet(TPM_CC commandCode) // IN: command code
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);

    // Only SET a bit if the corresponding command is implemented
    if (commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
    {
        // Can't audit shutdown
        if (commandCode != TPM_CC_Shutdown)
        {
            if (!TEST_BIT(commandIndex, gp.auditCommands))
            {
                // Set bit
                SET_BIT(commandIndex, gp.auditCommands);
                return TRUE;
            }
        }
    }
    // No change
    return FALSE;
}
```

8.1.3.4 CommandAuditClear()

This function will CLEAR the audit flag for a command. It will not CLEAR the audit flag for TPM_CC_SetCommandCodeAuditStatus().

This function is only used by TPM2_SetCommandCodeAuditStatus().

The actions in TPM2_SetCommandCodeAuditStatus() are expected to cause the changes to be saved to NV after it is setting and clearing bits.
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>command code audit status was changed</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>command code audit status was not changed</td>
</tr>
</tbody>
</table>

```c
BOOL CommandAuditClear(
    TPM_CC commandCode // IN: command code
)
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);
    // Do nothing if the command is not implemented
    if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
    {
        // The bit associated with TPM_CC_SetCommandCodeAuditStatus() cannot be cleared
        if(commandCode != TPM_CC_SetCommandCodeAuditStatus)
        {
            if(TEST_BIT(commandIndex, gp.auditCommands))
            {
                // Clear bit
                CLEAR_BIT(commandIndex, gp.auditCommands);
                return TRUE;
            }
        }
        // No change
        return FALSE;
    }
}
```

### 8.1.3.5 CommandAuditIsRequired()

This function indicates if the audit flag is SET for a command.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>command is audited</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>command is not audited</td>
</tr>
</tbody>
</table>

```c
BOOL CommandAuditIsRequired(
    COMMAND_INDEX commandIndex // IN: command index
)
{
    // Check the bit map. If the bit is SET, command audit is required
    return (TEST_BIT(commandIndex, gp.auditCommands));
}
```

### 8.1.3.6 CommandAuditCapGetCCList()

This function returns a list of commands that have their audit bit SET.

The list starts at the input `commandCode`. 

---

**Table and Code Analysis:**

- **Return Value** and **Meaning** tables are used to clearly define the outcomes of function calls and their implications.
- The `CommandAuditClear` function changes the audit status of a command, depending on whether the command is implemented and if the bit is associated with a specific audit status function.
- `CommandAuditIsRequired` checks if the audit bit is set for a given command index.
- `CommandAuditCapGetCCList` returns a list of commands with their audit bits set, starting from the specified `commandCode`.

---

**Code Snippets:**

```c
 BOOL CommandAuditClear(
     TPM_CC commandCode // IN: command code
 )
 {
     COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);
     // Do nothing if the command is not implemented
     if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
     {
         // The bit associated with TPM_CC_SetCommandCodeAuditStatus() cannot be cleared
         if(commandCode != TPM_CC_SetCommandCodeAuditStatus)
         {
             if (TEST_BIT(commandIndex, gp.auditCommands))
             {
                 // Clear bit
                 CLEAR_BIT(commandIndex, gp.auditCommands);
                 return TRUE;
             }
         }
         // No change
         return FALSE;
     }
 }
```
Return Value | Meaning
---|---
YES | if there are more command code available
NO | all the available command code has been returned

```c
TPMI_YES_NO
CommandAuditGetCCList(
  TPM_CC    commandCode, // IN: start command code
  UINT32    count, // IN: count of returned TPM_CC
  TPML_CC   *commandList // OUT: list of TPM_CC
)
{
  TPMI_YES_NO  more = NO;
  COMMAND_INDEX  commandIndex;

  // Initialize output handle list
  commandList->count = 0;

  // The maximum count of command we may return is MAX_CAP_CC
  if(count > MAX_CAP_CC) count = MAX_CAP_CC;

  // Find the implemented command that has a command code that is the same or
  // higher than the input
  // Collect audit commands
  for(commandIndex = GetClosestCommandIndex(commandCode);
      commandIndex != UNIMPLEMENTED_COMMAND_INDEX;
      commandIndex = GetNextCommandIndex(commandIndex))
  {
    if(CommandAuditIsRequired(commandIndex))
    {
      if(commandList->count < count)
      {
        // If we have not filled up the return list, add this command
        // code to its
        TPM_CC    cc = GET_ATTRIBUTE(s_ccAttr[commandIndex],
                                     TPMA_CC, commandIndex);
        if(IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V))
          cc += (1 << 29);
        commandList->commandCodes[commandList->count] = cc;
        commandList->count++;
      }
    }
    else
    {
      // If the return list is full but we still have command
      // available, report this and stop iterating
      more = YES;
      break;
    }
  }

  return more;
}
```

### 8.1.3.7 CommandAuditGetDigest

This command is used to create a digest of the commands being audited. The commands are processed in ascending numeric order with a list of TPM_CC being added to a hash. This operates as if all the audited command codes were concatenated and then hashed.

```c
void
CommandAuditGetDigest(
```

---

**Part 4: Supporting Routines**

**Trusted Platform Module Library**

**Family “2.0”**

**TCG Public Review**

**Level 00 Revision 01.50**

**Copyright © TCG 2006-2019**

**September 18, 2018**
TPM2B_DIGEST *digest    // OUT: command digest
}  
TPM_CC      commandCode;
COMMAND_INDEX commandIndex;
HASH_STATE  hashState;

// Start hash
digest->t.size = CryptHashStart(&hashState, gp.auditHashAlg);

// Add command code
for(commandIndex = 0; commandIndex < COMMAND_COUNT; commandIndex++)
{
    if(CommandAuditIsRequired(commandIndex))
    {
        commandCode = GetCommandCode(commandIndex);
        CryptDigestUpdateInt(&hashState, sizeof(commandCode), commandCode);
    }
}

// Complete hash
CryptHashEnd2B(&hashState, &digest->b);
return;
8.2 DA.c

8.2.1 Introduction

This file contains the functions and data definitions relating to the dictionary attack logic.

8.2.2 Includes and Data Definitions

```c
#define DA_C
#include "Tpm.h"
```

8.2.3 Functions

8.2.3.1 DAPreInstall_Init()

This function initializes the DA parameters to their manufacturer-default values. The default values are determined by a platform-specific specification.

This function should not be called outside of a manufacturing or simulation environment.

The DA parameters will be restored to these initial values by TPM2_Clear().

```c
void DAPreInstall_Init()
{
    gp.failedTries = 0;
    gp.maxTries = 3;
    gp.recoveryTime = 1000; // in seconds (~16.67 minutes)
    gp.lockoutRecovery = 1000; // in seconds
    gp.lockOutAuthEnabled = TRUE; // Use of lockoutAuth is enabled

    // Record persistent DA parameter changes to NV
    NV_SYNC_PERSISTENT(failedTries);
    NV_SYNC_PERSISTENT(maxTries);
    NV_SYNC_PERSISTENT(recoveryTime);
    NV_SYNC_PERSISTENT(lockoutRecovery);
    NV_SYNC_PERSISTENT(lockOutAuthEnabled);
    return;
}
```

8.2.3.2 DAStartup()

This function is called by TPM2_Startup() to initialize the DA parameters. In the case of Startup(CLEAR), use of lockoutAuth will be enabled if the lockout recovery time is 0. Otherwise, lockoutAuth will not be enabled until the TPM has been continuously powered for the lockoutRecovery time.

This function requires that NV be available and not rate limiting.

```c
void DAStartup(
    STARTUP_TYPE type       // IN: startup type
)
{
    NOT_REFERENCED(type);
    #if !ACCUMULATE_SELF_HEAL_TIMER
    _plat__TimerWasReset();
```
31     s_selfHealTimer = 0;
32     s_lockoutTimer = 0;
33 #else
34     if(_plat__TimerWasReset())
35     {
36         if(!NV_IS_ORDERLY)
37         {
38             // If shutdown was not orderly, then don't really know if go.time has
39             // any useful value so reset the timer to 0. This is what the tick
40             // was reset to
41             s_selfHealTimer = 0;
42             s_lockoutTimer = 0;
43         }
44     else
45         {
46             // If we know how much time was accumulated at the last orderly shutdown
47             // subtract that from the saved timer values so that they effectively
48             // have the accumulated values
49             s_selfHealTimer -= go.time;
50             s_lockoutTimer -= go.time;
51         }
52     } #endif
53
54     // For any Startup(), if lockoutRecovery is 0, enable use of lockoutAuth.
55     if(gp.lockoutRecovery == 0)
56     {
57         gp.lockOutAuthEnabled = TRUE;
58         // Record the changes to NV
59         NV_SYNC_PERSISTENT(lockOutAuthEnabled);
60     }
61
62     // If DA has not been disabled and the previous shutdown is not orderly
63     // failedTries is not already at its maximum then increment 'failedTries'
64     if(gp.recoveryTime != 0
65     && gp.failedTries < gp.maxTries
66     && !IS_ORDERLY(g_prevOrderlyState))
67     {
68         // If USE_DA_USED
69         gp.failedTries += g_daUsed;
70         g_daUsed = FALSE;
71     #else
72         gp.failedTries++;
73     #endif
74     // Record the change to NV
75     NV_SYNC_PERSISTENT(failedTries);
76     }
77     // Before Startup, the TPM will not do clock updates. At startup, need to
78     // do a time update which will do the DA update.
79     TimeUpdate();
80     return;
81 }
82

8.2.3.3  DARegisterFailure()

This function is called when a authorization failure occurs on an entity that is subject to dictionary-attack protection. When a DA failure is triggered, register the failure by resetting the relevant self-healing timer to the current time.

void
DARegisterFailure(
    TPM_HANDLE       handle     // IN: handle for failure
)
8.2.3.4 \texttt{DASelfHeal()}

This function is called to check if sufficient time has passed to allow decrement of \texttt{failedTries} or to re-enable use of \texttt{lockoutAuth}.

This function should be called when the time interval is updated.

```c
void
DASelfHeal(
    void
)
{
    // Regular authorization self healing logic
    // If no failed authorization tries, do nothing. Otherwise, try to
    // decrease failedTries
    if (gp.failedTries != 0) {
        // if recovery time is 0, DA logic has been disabled. Clear failed tries
        // immediately
        if (gp.recoveryTime == 0) {
            gp.failedTries = 0;
            // Update NV record
            NV_SYNC_PERSISTENT(failedTries);
        } else {
            UINT64 decreaseCount;
            #if 0 // Errata eliminates this code
                // In the unlikely event that failedTries should become larger than
                // maxTries
                if (gp.failedTries > gp.maxTries) {
                    gp.failedTries = gp.maxTries;
                }
            #endif
            // How much can failedTries be decreased
            // Cast \texttt{s_selfHealTimer} to an int in case it became negative at
            // startup
            decreaseCount = ((g_time - (INT64)s_selfHealTimer) / 1000)
                / gp.recoveryTime;
            if (gp.failedTries <= (UINT32)decreaseCount) {
                // should not set failedTries below zero
                gp.failedTries = 0;
            } else {
                gp.failedTries -= (UINT32)decreaseCount;
            }
            // the cast prevents overflow of the product
            s_selfHealTimer += (decreaseCount * (UINT64)gp.recoveryTime) * 1000;
            if (decreaseCount != 0) {
                // If there was a change to the failedTries, record the changes
                // to NV
                NV_SYNC_PERSISTENT(failedTries);
            }
        }
    }
}
```
// LockoutAuth self healing logic
// If lockoutAuth is enabled, do nothing. Otherwise, try to see if we
// may enable it
if(!gp.lockOutAuthEnabled)
{
    // if lockout authorization recovery time is 0, a reboot is required to
    // re-enable use of lockout authorization. Self-healing would not
    // apply in this case.
    if(gp.lockoutRecovery != 0)
    {
        if(((g_time - (INT64)s_lockoutTimer) / 1000) >= gp.lockoutRecovery)
            gp.lockOutAuthEnabled = TRUE;
        // Record the changes to NV
        NV_SYNC_PERSISTENT(lockOutAuthEnabled);
    }
    return;
}
8.3 Hierarchy.c

8.3.1 Introduction

This file contains the functions used for managing and accessing the hierarchy-related values.

8.3.2 Includes

```c
#include "Tpm.h"
```

8.3.3 Functions

8.3.3.1 HierarchyPreInstall()

This function performs the initialization functions for the hierarchy when the TPM is simulated. This function should not be called if the TPM is not in a manufacturing mode at the manufacturer, or in a simulated environment.

```c
void
HierarchyPreInstall_Init(
    void
) {
    // Allow lockout clear command
    gp.disableClear = FALSE;

    // Initialize Primary Seeds
    gp.EPSeed.t.size = sizeof(gp.EPSeed.t.buffer);
    gp.SPSeed.t.size = sizeof(gp.SPSeed.t.buffer);
    gp.PPSeed.t.size = sizeof(gp.PPSeed.t.buffer);
    #if (defined USE_PLATFORM_EPS) && (USE_PLATFORM_EPS != NO)
        plat__GetEPS(gp.EPSeed.t.size, gp.EPSeed.t.buffer);
    #else
        CryptRandomGenerate(gp.EPSeed.t.size, gp.EPSeed.t.buffer);
    #endif
    CryptRandomGenerate(gp.SPSeed.t.size, gp.SPSeed.t.buffer);
    CryptRandomGenerate(gp.PPSeed.t.size, gp.PPSeed.t.buffer);

    // Initialize owner, endorsement and lockout authorization
    gp.ownerAuth.t.size = 0;
    gp.endorsementAuth.t.size = 0;
    gp.lockoutAuth.t.size = 0;

    // Initialize owner, endorsement, and lockout policy
    gp.ownerAlg = TPM_ALG_NULL;
    gp.ownerPolicy.t.size = 0;
    gp.endorsementAlg = TPM_ALG_NULL;
    gp.endorsementPolicy.t.size = 0;
    gp.lockoutAlg = TPM_ALG_NULL;
    gp.lockoutPolicy.t.size = 0;

    // Initialize ehProof, shProof and phProof
    gp.phProof.t.size = sizeof(gp.phProof.t.buffer);
    gp.shProof.t.size = sizeof(gp.shProof.t.buffer);
    gp.ehProof.t.size = sizeof(gp.ehProof.t.buffer);
    CryptRandomGenerate(gp.phProof.t.size, gp.phProof.t.buffer);
    CryptRandomGenerate(gp.shProof.t.size, gp.shProof.t.buffer);
    CryptRandomGenerate(gp.ehProof.t.size, gp.ehProof.t.buffer);

    // Write hierarchy data to NV
```
8.3.3.2 HierarchyStartup()

This function is called at TPM2_Startup() to initialize the hierarchy related values.

```c
void HierarchyStartup(
    STARTUP_TYPE     type     // IN: start up type
)
{
    // phEnable is SET on any startup
    g_phEnable = TRUE;

    // Reset platformAuth, platformPolicy; enable SH and EH at TPM_RESET and
    // TPM_RESTART
    if(type != SU_RESUME)
    {
        gc.platformAuth.t.size = 0;
        gc.platformPolicy.t.size = 0;

        // enable the storage and endorsement hierarchies and the platformNV
        gc.shEnable = gc.ehEnable = gc.phEnableNV = TRUE;
    }

    // nullProof and nullSeed are updated at every TPM_RESET
    if((type != SU_RESTART) && (type != SU_RESUME))
    {
        gr.nullProof.t.size = sizeof(gr.nullProof.t.buffer);
        CryptRandomGenerate(gr.nullProof.t.size, gr.nullProof.t.buffer);
        gr.nullSeed.t.size = sizeof(gr.nullSeed.t.buffer);
        CryptRandomGenerate(gr.nullSeed.t.size, gr.nullSeed.t.buffer);
    }

    return;
}
```

8.3.3.3 HierarchyGetProof()

This function finds the proof value associated with a hierarchy. It returns a pointer to the proof value.

```c
TPM2B_PROOF * HierarchyGetProof(
    TPMI_RH_HIERARCHY    hierarchy    // IN: hierarchy constant
)
```
8.3.3.4 HierarchyGetPrimarySeed()

This function returns the primary seed of a hierarchy.

```c
TPM2B_SEED * HierarchyGetPrimarySeed(
    TPM2B_SEED *seed = NULL;
switch (hierarchy)
    { case TPM_RH_PLATFORM:
        seed = &gp.PPSeed;
        break;
    case TPM_RH_OWNER:
        seed = &gp.SPSeed;
        break;
    case TPM_RH_ENDORSEMENT:
        seed = &gp.EPSeed;
        break;
    default:
        FAIL(FATAL_ERROR_INTERNAL);
        break;
    }
    return seed;
}
```

8.3.3.5 HierarchyIsEnabled()

This function checks to see if a hierarchy is enabled.
NOTE: The TPM_RH_NULL hierarchy is always enabled.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>hierarchy is enabled</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>hierarchy is disabled</td>
</tr>
</tbody>
</table>

```c
BOOL HierarchyIsEnabled(
    TPMI_RH_HIERARCHY hierarchy    // IN: hierarchy
) {
    BOOL enabled = FALSE;

    switch(hierarchy) {
        case TPM_RH_PLATFORM:
            enabled = g_phEnable;
            break;
        case TPM_RH_OWNER:
            enabled = gc.shEnable;
            break;
        case TPM_RH_ENDORSEMENT:
            enabled = gc.ehEnable;
            break;
        case TPM_RH_NULL:
            enabled = TRUE;
            break;
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }

    return enabled;
}
```
8.4  NvDynamic.c

8.4.1  Introduction

The NV memory is divided into two areas: dynamic space for user-defined NV indexes and evict objects, and reserved space for TPM persistent and state save data.

The entries in dynamic space are a linked list of entries. Each entry has, as its first field, a size. If the size field is zero, it marks the end of the list.

An Index allocation will contain an NV_INDEX structure. If the Index does not have the orderly attribute, the NV_INDEX is followed immediately by the NV data.

An evict object entry contains a handle followed by an OBJECT structure. This results in both the Index and Evict Object having an identifying handle as the first field following the size field.

When an Index has the orderly attribute, the data is kept in RAM. This RAM is saved to backing store in NV memory on any orderly shutdown. The entries in orderly memory are also a linked list using a size field as the first entry.

The attributes of an orderly index are maintained in RAM memory in order to reduce the number of NV writes needed for orderly data. When an orderly index is created, an entry is made in the dynamic NV memory space that holds the Index authorizations (authPolicy and authValue) and the size of the data. This entry is only modified if the authValue of the index is changed. The more volatile data of the index is kept in RAM. When an orderly Index is created or deleted, the RAM data is copied to NV backing store so that the image in the backing store matches the layout of RAM. In normal operation, the RAM data is also copied on any orderly shutdown. In normal operation, the only other reason for writing to the backing store for RAM is when a counter is first written (TPMA_NV_WRITTEN changes from CLEAR to SET) or when a counter "rolls over."

Static space contains items that are individually modifiable. The values are in the gp PERSISTEND_DATA structure in RAM and mapped to locations in NV.

8.4.2  Includes, Defines and Data Definitions

```c
#define NV_C
#include "Tpm.h"
#include "PlatformData.h"
```

8.4.3  Local Functions

8.4.3.1  NvNext()

This function provides a method to traverse every data entry in NV dynamic area.

To begin with, parameter iter should be initialized to NV_REF_INIT indicating the first element. Every time this function is called, the value in iter would be adjusted pointing to the next element in traversal. If there is no next element, iter value would be 0. This function returns the address of the 'data entry' pointed by the iter. If there is no more element in the set, a 0 value is returned indicating the end of traversal.

```c
static NV_REF
NvNext(
    NV_REF *iter,          // IN/OUT: the list iterator
    TPM_HANDLE *handle    // OUT: the handle of the next item.
)
{
    NV_REF               currentAddr;
    NV_ENTRY_HEADER     header;
```
8.4.3.2 NvNextByType()

This function returns a reference to the next NV entry of the desired type.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>end of list</td>
</tr>
<tr>
<td>!= 0</td>
<td>the next entry of the indicated type</td>
</tr>
</tbody>
</table>

```c
static NV_REF
NvNextByType(
    TPM_HANDLE *handle,  // OUT: the handle of the found type
    NV_REF *iter,       // IN: the iterator
    TPM_HT type         // IN: the handle type to look for
) {
    NV_REF addr;
    TPM_HANDLE nvHandle;
    //
    while((addr = NvNext(iter, &nvHandle)) != 0)
        {
            // addr: the address of the location containing the handle of the value
            // iter: the next location.
            if(HandleGetType(nvHandle) == type)
                break;
        }
    if(handle != NULL)
        *handle = nvHandle;
    return addr;
}
```

8.4.3.3 NvNextIndex()

This function returns the reference to the next NV Index entry. A value of 0 indicates the end of the list.
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>end of list</td>
</tr>
<tr>
<td>!= 0</td>
<td>the next reference</td>
</tr>
</tbody>
</table>

```c
#define NvNextIndex(handle, iter)  
    NvNextByType(handle, iter, TPM_HT_NV_INDEX)

8.4.3.4 NvNextEvict()

This function returns the offset in NV of the next evict object entry. A value of 0 indicates the end of the list.

```c
#define NvNextEvict(handle, iter)  
    NvNextByType(handle, iter, TPM_HT_PERSISTENT)

8.4.3.5 NvGetEnd()

Function to find the end of the NV dynamic data list

```c
static NV_REF
NvGetEnd()
{
    void
}
    
    NV_REF iter = NV_REF_INIT;
    NV_REF currentAddr;
    
    // Scan until the next address is 0
    while((currentAddr = NvNext(&iter, NULL)) != 0);
    return iter;
}

8.4.3.6 NvGetFreeBytes

This function returns the number of free octets in NV space.

```c
static UINT32
NvGetFreeBytes()
{
    void
    
    // This does not have an overflow issue because NvGetEnd() cannot return a value
    // that is larger than s_evictNvEnd. This is because there is always a 'stop'
    // word in the NV memory that terminates the search for the end before the
    // value can go past s_evictNvEnd.
    return s_evictNvEnd - NvGetEnd();
}

8.4.3.7 NvTestSpace()

This function will test if there is enough space to add a new entity.


<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>space available</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>no enough space</td>
</tr>
</tbody>
</table>

```c
static BOOL NvTestSpace(
    UINT32 size, // IN: size of the entity to be added
    BOOL isIndex, // IN: TRUE if the entity is an index
    BOOL isCounter // IN: TRUE if the index is a counter
)
{
    UINT32 remainBytes = NvGetFreeBytes();
    UINT32 reserved = sizeof(UINT32) // size of the forward pointer
                     + sizeof(NV_LIST_TERMINATOR);

    // Do a compile time sanity check on the setting for NV_MEMORY_SIZE
    #if NV_MEMORY_SIZE < 1024
        #error "NV_MEMORY_SIZE probably isn't large enough"
    #endif

    // For NV Index, need to make sure that we do not allocate an Index if this
    // would mean that the TPM cannot allocate the minimum number of evict
    // objects.
    if(isIndex)
    {
        // Get the number of persistent objects allocated
        UINT32 persistentNum = NvCapGetPersistentNumber();

        // If we have not allocated the requisite number of evict objects, then we
        // need to reserve space for them.
        // NOTE: some of this is not written as simply as it might seem because
        // the values are all unsigned and subtracting needs to be done carefully
        // so that an underflow doesn't cause problems.
        if(persistentNum < MIN_EVICT_OBJECTS)
            reserved += (MIN_EVICT_OBJECTS - persistentNum) * NV_EVICT_OBJECT_SIZE;
    }
    // If this is not an index or is not a counter, reserve space for the
    // required number of counter indexes
    if(!isIndex || !isCounter)
    {
        // Get the number of counters
        UINT32 counterNum = NvCapGetCounterNumber();

        // If the required number of counters have not been allocated, reserved
        // space for the extra needed counters
        if(counterNum < MIN_COUNTER_INDICES)
            reserved += (MIN_COUNTER_INDICES - counterNum) * NV_INDEX_COUNTER_SIZE;
    }

    // Check that the requested allocation will fit after making sure that there
    // will be no chance of overflow
    return ((reserved < remainBytes)
              && (size <= remainBytes)
              && (size + reserved <= remainBytes));
}
```

### 8.4.3.8 NvWriteNvListEnd()

Function to write the list terminator.
137 }
138 {
139     // Marker is initialized with zeros
140     BYTE listEndMarker[sizeof(NV_LIST_TERMINATOR)] = {0};
141     UINT64 maxCount = NvReadMaxCount();
142     // This is a constant check that can be resolved at compile time.
143     cAssert(sizeof(UINT64) <= sizeof(NV_LIST_TERMINATOR) - sizeof(UINT32));
144     // Copy the maxCount value to the marker buffer
145     MemoryCopy(&listEndMarker[sizeof(UINT32)], &maxCount, sizeof(UINT64));
146     pAssert(end + sizeof(NV_LIST_TERMINATOR) <= s_evictNvEnd);
147     // Write it to memory
148     NvWrite(end, sizeof(NV_LIST_TERMINATOR), &listEndMarker);
149     return end + sizeof(NV_LIST_TERMINATOR);
150 }
151
8.4.3.9 NvAdd()
152
This function adds a new entity to NV.
This function requires that there is enough space to add a new entity (i.e., that NvTestSpace() has been called and the available space is at least as large as the required space).
The totalSize will be the size of entity. If a handle is added, this function will increase the size accordingly.

154 static TPM_RC
155 NvAdd(156    UINT32 totalSize, // IN: total size needed for this entity For
157        evict object, totalSize is the same as
158        // bufferSize. For NV Index, totalSize is
159        // bufferSize plus index data size
160    UINT32 bufferSize, // IN: size of initial buffer
161    TPM_HANDLE handle, // IN: optional handle
162    BYTE *entity // IN: initial buffer
163 )
164 {
165     // RETURN_IF_NV_IS_NOT_AVAILABLE;
166     // Get the end of data list
167     newAddr = NvGetEnd();
168     // Step over the forward pointer
169     nextAddr = newAddr + sizeof(UINT32);
170     // Optionally write the handle. For indexes, the handle is TPM_RH_UNASSIGNED
171     // so that the handle in the nvIndex is used instead of writing this value
172     if(handle != TPM_RH_UNASSIGNED)
173     {
174         NvWrite((UINT32)nextAddr, sizeof(TPM_HANDLE), &handle);
175         nextAddr += sizeof(TPM_HANDLE);
176     }
177     // Write entity data
178     NvWrite((UINT32)nextAddr, bufferSize, entity);
179     // Advance the pointer by the amount of the total
180     nextAddr += totalSize;
181     // Finish by writing the link value
182     // Write the next offset (relative addressing)
8.4.3.10 NvDelete()

This function is used to delete an NV Index or persistent object from NV memory.

```c
static TPM_RC
NvDelete(NV_REF entityRef) // IN: reference to entity to be deleted
{
    UINT32 entrySize;
    // adjust entryAddr to back up and point to the forward pointer
    NV_REF entryRef = entityRef - sizeof(UINT32);
    NV_REF endRef = NvGetEnd();
    NV_REF nextAddr; // address of the next entry
    // RETURN_IF_NV_IS_NOT_AVAILABLE;
    // Get the offset of the next entry. That is, back up and point to the size
    // field of the entry
    NvRead(&entrySize, entryRef, sizeof(UINT32));
    // The next entry after the one being deleted is at a relative offset
    // from the current entry
    nextAddr = entryRef + entrySize;
    // If this is not the last entry, move everything up
    if(nextAddr < endRef)
    {
        pAssert(nextAddr > entryRef);
        _plat__NvMemoryMove(nextAddr, entryRef,
                              (endRef - nextAddr));
    }
    // The end of the used space is now moved up by the amount of space we just
    // reclaimed
    endRef -= entrySize;
    // Write the end marker, and make the new end equal to the first byte after
    // the just added end value. This will automatically update the NV value for
    // maxCounter.
    // NOTE: This is the call that sets flag to cause NV to be updated
    endRef = NvWriteNvListEnd(endRef);
    // Clear the reclaimed memory
    _plat__NvMemoryClear(endRef, entrySize);
    return TPM_RC_SUCCESS;
}
```
8.4.4 RAM-based NV Index Data Access Functions

8.4.4.1 Introduction

The data layout in ram buffer is \( (\text{size of} \ NV\_\text{handle()} + \text{attributes} + \text{data} \ NV\_\text{handle()}, \text{attributes}, \text{data}) \) for each NV Index data stored in RAM.

NV storage associated with orderly data is updated when a NV Index is added but NOT when the data or attributes are changed. Orderly data is only updated to NV on an orderly shutdown (TPM2_Shutdown()).

8.4.4.2 NvRamNext()

This function is used to iterate through the list of Ram Index values. *iter needs to be initialized by calling

```c
static NV_RAM_REF
NvRamNext(
    NV_RAM_REF *iter, // IN/OUT: the list iterator
    TPM_HANDLE *handle // OUT: the handle of the next item.
)
```

```c
    NV_RAM_REF           currentAddr;
    NV_RAM_HEADER        header;
    // If iterator is at the beginning of list
    if(*iter == NV_RAM_REF_INIT)
    {
        // Initialize iterator
        *iter = &s_indexOrderlyRam[0];
    }
    // if we are going to return what the iter is currently pointing to...
    currentAddr = *iter;
    // If iterator reaches the end of NV space, then don't advance and return
    // that we are at the end of the list. The end of the list occurs when
    // we don't have space for a size and a handle
    if(currentAddr + sizeof(NV_RAM_HEADER) > RAM_ORDERLY_END)
        return NULL;
    // read the header of the next entry
    MemoryCopy(&header, currentAddr, sizeof(NV_RAM_HEADER));
    // if the size field is zero, then we have hit the end of the list
    if(header.size == 0)
        // leave the *iter pointing at the end of the list
        return NULL;
    // advance the header by the size of the entry
    *iter = currentAddr + header.size;
    // pAssert(*iter <= RAM_ORDERLY_END);
    if(handle != NULL)
        *handle = header.handle;
    return currentAddr;
```

8.4.4.3 NvRamGetEnd()

This routine performs the same function as NvGetEnd() but for the RAM data.

```c
static NV_RAM_REF
NvRamGetEnd(
    void
)
```
8.4.4.4 NvRamTestSpaceIndex()

This function indicates if there is enough RAM space to add a data for a new NV Index.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>space available</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>no enough space</td>
</tr>
</tbody>
</table>

```c
static BOOL NvRamTestSpaceIndex(
    UINT32 size);  // IN: size of the data to be added to RAM
```

8.4.4.5 NvRamGetIndex()

This function returns the offset of NV data in the RAM buffer

This function requires that NV Index is in RAM. That is, the index must be known to exist.

```c
static NV_RAM_REF NvRamGetIndex(
    TPMI_RH_NV_INDEX handle);  // IN: NV handle
```

8.4.4.6 NvUpdateIndexOrderlyData()

This function is used to cause an update of the orderly data to the NV backing store.

```c
void NvUpdateIndexOrderlyData();
```
8.4.4.7 NvAddRAM()

This function adds a new data area to RAM.

This function requires that enough free RAM space is available to add the new data.

This function should be called after the NV Index space has been updated and the index removed. This insures that NV is available so that checking for NV availability is not required during this function.

```c
static void
NvAddRAM(
    TPMS_NV_PUBLIC *index     // IN: the index descriptor
)
{
    NV_RAM_HEADER header;
    NV_RAM_REF end = NvRamGetEnd();
    // header.size = sizeof(NV_RAM_HEADER) + index->dataSize;
    header.handle = index->nvIndex;
    MemoryCopy(&header.attributes, &index->attributes, sizeof(TPMA_NV));
    pAssert(ORDERLY_RAM_ADDRESS_OK(end, header.size));
    // Copy the header to the memory
    MemoryCopy(end, &header, sizeof(NV_RAM_HEADER));
    // Clear the data area (just in case)
    MemorySet(end + sizeof(NV_RAM_HEADER), 0, index->dataSize);
    // Step over this new entry
    end += header.size;
    // If the end marker will fit, add it
    if(end + sizeof(UINT32) < RAM_ORDERLY_END)
        MemorySet(end, 0, sizeof(UINT32));
    // Write reserved RAM space to NV to reflect the newly added NV Index
    SET_NV_UPDATE(UT_ORDERLY);
    return;
}
```

8.4.4.8 NvDeleteRAM()

This function is used to delete a RAM-backed NV Index data area. The space used by the entry are overwritten by the contents of the Index data that comes after (the data is moved up to fill the hole left by removing this index. The reclaimed space is cleared to zeros. This function assumes the data of NV Index exists in RAM.

This function should be called after the NV Index space has been updated and the index removed. This insures that NV is available so that checking for NV availability is not required during this function.

```c
static void
NvDeleteRAM(
    TPMI_RH_NV_INDEX handle     // IN: NV handle
)
{
    NV_RAM_REF nodeAddress;
    NV_RAM_REF nextNode;
```
UINT32 size;

NV_RAM_REF lastUsed = NvRamGetEnd();

// nodeAddress = NvRamGetIndex(handle);

pAssert(nodeAddress != 0);

// Get node size
MemoryCopy(&size, nodeAddress, sizeof(size));

// Get the offset of next node
nextNode = nodeAddress + size;

// Copy the data
MemoryCopy(nodeAddress, nextNode, lastUsed - nextNode);

// Clear out the reclaimed space
MemorySet(lastUsed - size, 0, size);

// Write reserved RAM space to NV to reflect the newly delete NV Index
SET_NV_UPDATE(UT_ORDERLY);

return;
}

8.4.4.9 NvReadIndex()

This function is used to read the NV Index NV_INDEX. This is used so that the index information can be compressed and only this function would be needed to decompress it. Mostly, compression would only be able to save the space needed by the policy.

void
NvReadNvIndexInfo(
    NV_REF ref,   // IN: points to NV where index is located
    NV_INDEX *nvIndex    // OUT: place to receive index data
)
{
    pAssert(nvIndex != NULL);

    NvRead(nvIndex, ref, sizeof(NV_INDEX));

    return;
}

8.4.4.10 NvReadObject()

This function is used to read a persistent object. This is used so that the object information can be compressed and only this function would be needed to uncompress it.

void
NvReadObject(
    NV_REF ref,   // IN: points to NV where index is located
    OBJECT *object    // OUT: place to receive the object data
)
{
    NvRead(object, (ref + sizeof(TPM_HANDLE)), sizeof(OBJECT));

    return;
}

8.4.4.11 NvFindEvict()

This function will return the NV offset of an evict object
### Return Value | Meaning
---|---
0 | evict object not found
!= 0 | offset of evict object

```c
412  static NV_REF
413  NvFindEvict(
414      TPM_HANDLE  nvHandle,
415      OBJECT      *object
416  )
417  {
418      NV_REF      found = NvFindHandle(nvHandle);
419      // If we found the handle and the request included an object pointer, fill it in
420      if(found != 0 && object != NULL)
421          NvReadObject(found, object);
422      return found;
423  }
```

#### 8.4.4.12 NvIndexIsDefined()

See if an index is already defined

```c
425  BOOL
426  NvIndexIsDefined(
427      TPM_HANDLE       nvHandle   // IN: Index to look for
428  )
429  {
430      return (NvFindHandle(nvHandle) != 0);
431  }
```

#### 8.4.4.13 NvConditionallyWrite()

Function to check if the data to be written has changed and write it if it has

```c
432  static TPM_RC
433  NvConditionallyWrite(
434      NV_REF          entryAddr,  // IN: stating address
435      UINT32         size,      // IN: size of the data to write
436      void           *data      // IN: the data to write
437  )
438  {
439      // If the index data is actually changed, then a write to NV is required
440      if(_plat__NvIsDifferent(entryAddr, size, data))
441      {
442          // Write the data if NV is available
443          if(g_NvStatus == TPM_RC_SUCCESS)
444          {
445              NvWrite(entryAddr, size, data);
446          }
447          return g_NvStatus;
448      }
449      return TPM_RC_SUCCESS;
450  }
```
8.4.4.14 NvReadNvIndexAttributes()

This function returns the attributes of an NV Index.

```c
static TPMA_NV
NvReadNvIndexAttributes(
    NV_REF           locator    // IN: reference to an NV index
) {
    TPMA_NV kayattributes;
    // NvRead(&attributes,
    locator + offsetof(NV_INDEX, publicArea.attributes),
    sizeof(TPMA_NV));
    return attributes;
}
```

8.4.4.15 NvReadRamIndexAttributes()

This function returns the attributes from the RAM header structure. This function is used to deal with the fact that the header structure is only byte aligned.

```c
static TPMA_NV
NvReadRamIndexAttributes(
    NV_RAM_REF       ref       // IN: pointer to a NV_RAM_HEADER
) {
    TPMA_NV atributes;
    // MemoryCopy(&attributes, ref + offsetof(NV_RAM_HEADER, attributes),
    sizeof(TPMA_NV));
    return attributes;
}
```

8.4.4.16 NvWriteNvIndexAttributes()

This function is used to write just the attributes of an index to NV.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>NV is rate limiting so retry</td>
</tr>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>NV is not available</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
NvWriteNvIndexAttributes(
    NV_REF           locator,    // IN: location of the index
    TPMA_NV attributes // IN: attributes to write
) {
    return NvConditionallyWrite(
        locator + offsetof(NV_INDEX, publicArea.attributes),
        sizeof(TPMA_NV),
        &attributes);
}
```

8.4.4.17 NvWriteRamIndexAttributes()

This function is used to write the index attributes into an unaligned structure.
static void NvWriteRamIndexAttributes(
    NV_RAM_REF ref,       // IN: address of the header
    TPMA_NV attributes    // IN: the attributes to write
) {
    MemoryCopy(ref + offsetof(NV_RAM_HEADER, attributes), &attributes, sizeof(TPMA_NV));
    return;
}

8.4.5 Externally Accessible Functions

8.4.5.1 NvIsPlatformPersistentHandle()

This function indicates if a handle references a persistent object in the range belonging to the platform.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>handle references a platform persistent object and may reference an owner persistent object either</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>handle does not reference platform persistent object</td>
</tr>
</tbody>
</table>

BOOL NvIsPlatformPersistentHandle(
    TPM_HANDLE handle       // IN: handle
) {
    return (handle >= PLATFORM_PERSISTENT && handle <= PERSISTENT_LAST);
}

8.4.5.2 NvIsOwnerPersistentHandle()

This function indicates if a handle references a persistent object in the range belonging to the owner.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>handle is owner persistent handle</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>handle is not owner persistent handle and may not be a persistent handle at all</td>
</tr>
</tbody>
</table>

BOOL NvIsOwnerPersistentHandle(
    TPM_HANDLE handle       // IN: handle
) {
    return (handle >= PERSISTENT_FIRST && handle < PLATFORM_PERSISTENT);
}

8.4.5.3 NvIndexIsAccessible()

This function validates that a handle references a defined NV Index and that the Index is currently accessible.
### Error Returns

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HANDLE</td>
<td>the handle points to an undefined NV Index If shEnable is CLEAR, this</td>
</tr>
<tr>
<td></td>
<td>would include an index created using ownerAuth. If phEnableNV is</td>
</tr>
<tr>
<td></td>
<td>CLEAR, this would include index created using platformAuth</td>
</tr>
<tr>
<td>TPM_RC_NV_READLOCKED</td>
<td>Index is present but locked for reading and command does not write</td>
</tr>
<tr>
<td>TPM_RC_NV_WRITELOCKED</td>
<td>Index is present but locked for writing and command writes to the</td>
</tr>
</tbody>
</table>

```c
TPM_RC
NvIndexIsAccessible(
    TPMI_RH_NV_INDEX handle     // IN: handle
) {
    NV_INDEX *nvIndex = NvGetIndexInfo(handle, NULL);
    //
    if(nvIndex == NULL)
        // If index is not found, return TPM_RC_HANDLE
        return TPM_RC_HANDLE;
    if(gc.shEnable == FALSE || gc.phEnableNV == FALSE)
        {
            // if shEnable is CLEAR, an ownerCreate NV Index should not be
            // indicated as present
            if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, PLATFORMCREATE))
                {
                if(gc.shEnable == FALSE)
                    return TPM_RC_HANDLE;
                }
            // if phEnableNV is CLEAR, a platform created Index should not
            // be visible
            else if(gc.phEnableNV == FALSE)
                return TPM_RC_HANDLE;
        }
    #if 0 // Writelock test for debug
    // If the Index is write locked and this is an NV Write operation...
    if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITELOCKED)
        && IsWriteOperation(commandIndex))
        {
            // then return a locked indication unless the command is TPM2_NV_WriteLock
            if(GetCommandCode(commandIndex) != TPM_CC_NV_WriteLock)
                return TPM_RC_NV_LOCKED;
            return TPM_RC_SUCCESS;
    }
    #endif
    #if 0 // Readlock Test for debug
    // If the Index is read locked and this is an NV Read operation...
    if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, READLOCKED)
        && IsReadOperation(commandIndex))
        {
            // then return a locked indication unless the command is TPM2_NV_ReadLock
            if(GetCommandCode(commandIndex) != TPM_CC_NV_ReadLock)
                return TPM_RC_NV_LOCKED;
        }
    #endif
    // NV Index is accessible
    return TPM_RC_SUCCESS;
}
```

#### 8.4.5.4  NvGetEvictObject()

This function is used to dereference an evict object handle and get a pointer to the object.
<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HANDLE</td>
<td>the handle does not point to an existing persistent object</td>
</tr>
</tbody>
</table>

```c
TPM_RC
NvGetEvictObject(
    TPM_HANDLE handle, // IN: handle
    OBJECT *object   // OUT: object data
)
{
    NV_REF entityAddr; // offset points to the entity
    // Find the address of evict object and copy to object
    entityAddr = NvFindEvict(handle, object);
    // whether there is an error or not, make sure that the evict
    // status of the object is set so that the slot will get freed on exit
    // Must do this after NvFindEvict loads the object
    object->attributes.evict = SET;
    // If handle is not found, return an error
    if(entityAddr == 0)
        return TPM_RC_HANDLE;
    return TPM_RC_SUCCESS;
}
```

### 8.4.5.5 NvIndexCacheInit()

Function to initialize the Index cache

```c
void
NvIndexCacheInit(
    void
)
{
    s_cachedNvRef = NV_REF_INIT;
    s_cachedNvRamRef = NV_RAM_REF_INIT;
    s_cachedNvIndex.publicArea.nvIndex = TPM_RH_UNASSIGNED;
    return;
}
```

### 8.4.5.6 NvGetIndexData()

This function is used to access the data in an NV Index. The data is returned as a byte sequence.

This function requires that the NV Index be defined, and that the required data is within the data range. It also requires that TPMA_NV_WRITTEN of the Index is SET.

```c
void
NvGetIndexData(
    NV_INDEX *nvIndex, // IN: the in RAM index descriptor
    NV_REF locator,   // IN: where the data is located
    UINT32 offset,    // IN: offset of NV data
    UINT16 size,      // IN: size of NV data
    void *data        // OUT: data buffer
)
{
    TPMA_NV nvAttributes;
    // pAssert(nvIndex != NULL);
    nvAttributes = nvIndex->publicArea.attributes;
```
pAssert(IS_ATTRIBUTE(nvAttributes, TPMA_NV, WRITTEN));
if (IS_ATTRIBUTE(nvAttributes, TPMA_NV, ORDERLY)) {
    // Get data from RAM buffer
    NV_RAM_REF ramAddr = NvRamGetIndex(nvIndex->publicArea.nvIndex);
    pAssert(ramAddr != 0 && (size <= ((NV_RAM_HEADER *)ramAddr)->size - sizeof(NV_RAM_HEADER) - offset));
    MemoryCopy(data, ramAddr + sizeof(NV_RAM_HEADER) + offset, size);
} else {
    // Validate that read falls within range of the index
    pAssert(offset <= nvIndex->publicArea.dataSize && size <= (nvIndex->publicArea.dataSize - offset));
    NvRead(data, locator + sizeof(NV_INDEX) + offset, size);
} return;

8.4.5.7 NvGetUINT64Data()
Get data in integer format of a bit or counter NV Index.
This function requires that the NV Index is defined and that the NV Index previously has been written.

UINT64 NvGetUINT64Data(
    NV_INDEX *nvIndex,       // IN: the in RAM index descriptor
    NV_REF locator           // IN: where index exists in NV
) {
    UINT64                intVal;
    // Read the value and convert it to internal format
    NvGetIndexData(nvIndex, locator, 0, 8, &intVal);
    return BYTE_ARRAY_TO_UINT64(((BYTE *)&intVal));
}

8.4.5.8 NvWriteIndexAttributes()
This function is used to write just the attributes of an index.

Error Returns | Meaning
---|---
TPM_RC_NV_RATE | NV is rate limiting so retry
TPM_RC_NV_UNAVAILABLE | NV is not available

TPM_RC NvWriteIndexAttributes(
    TPM_HANDLE handle,
    NV_REF locator,       // IN: location of the index
    TPMA_NV attributes    // IN: attributes to write
) {
    TPM_RC              result;
    //
    if (IS_ATTRIBUTE(attributes, TPMA_NV, ORDERLY)) {
        NV_RAM_REF ram = NvRamGetIndex(handle);
        NvWriteRamIndexAttributes(ram, attributes);
    }
8.4.5.9 NvWriteIndexAuth()

This function is used to write the authValue of an index. It is used by TPM2_NV_ChangeAuth()

```
8.4.5.10 NvGetIndexInfo()

This function loads the nvIndex Info into the NV cache and returns a pointer to the NV_INDEX. If the returned value is zero, the index was not found. The locator parameter, if not NULL, will be set to the offset in NV of the Index (the location of the handle of the Index).

This function will set the index cache. If the index is orderly, the attributes from RAM are substituted for the attributes in the cached index

```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>NV is rate limiting so retry</td>
</tr>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>NV is not available</td>
</tr>
</tbody>
</table>

```

TPM_RC

NvWriteIndexAuth()

```

NvGetIndexInfo()
8.4.5.11 NvWriteIndexData()

This function is used to write NV index data. It is intended to be used to update the data associated with the default index.

This function requires that the NV Index is defined, and the data is within the defined data range for the index.

Index data is only written due to a command that modifies the data in a single index. There is no case where changes are made to multiple indexes data at the same time. Multiple attributes may be change but not multiple index data. This is important because we will normally be handling the index for which we have the cached pointer values.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>NV is rate limiting so retry</td>
</tr>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>NV is not available</td>
</tr>
</tbody>
</table>

```c
TPM_RC
NvWriteIndexData(
    NV_INDEX        *nvIndex,  // IN: the description of the index
    UINT32           offset,  // IN: offset of NV data
    UINT32           size,    // IN: size of NV data
    void             *data    // IN: data buffer
)

    TPM_RC               result = TPM_RC_SUCCESS;
    //
    pAssert(nvIndex != NULL);
    // Make sure that this is dealing with the 'default' index.
    // Note: it is tempting to change the calling sequence so that the 'default' is
    // presumed.
    pAssert(nvIndex->publicArea.nvIndex == s_cachedNvIndex.publicArea.nvIndex);
    // Validate that write falls within range of the index
    pAssert(offset <= nvIndex->publicArea dataSize
            && size <= (nvIndex->publicArea dataSize - offset));
    // Update TPMA_NV_WRITTEN bit if necessary
    if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))
        {
            // Update the in memory version of the attributes
            SET_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN);
            // If this is not orderly, then update the NV version of
            // the attributes
            if(!IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, ORDERLY))
                {
                    result = NvWriteNvIndexAttributes(s_cachedNvRef,
```
8.4.5.12  NvWriteUINT64Data()

This function to write back a UINT64 value. The various UINT64 values (bits, counters, and PINs()) are kept in canonical format but manipulate in native format. This takes a native format value converts it and saves it back as in canonical format.

This function will return the value from NV or RAM depending on the type of the index (orderly or not)
BYTE bytes[8];
UINT64_TO_BYTE_ARRAY(intValue, bytes);

// return NvWriteIndexData(nvIndex, 0, 8, &bytes);
}

8.4.5.13 NvGetIndexName()

This function computes the Name of an index. The name buffer receives the bytes of the Name and the return value is the number of octets in the Name.

This function requires that the NV Index is defined.

TPM2B_NAME *
NvGetIndexName(
    NV_INDEX *nvIndex,  // IN: the index over which the name is to be
    TPM2B_NAME *name   // OUT: name of the index
)
{
    TPM2B_NAME *name;
    TArray byte[];
    UINT16 nameSize;
    BYTE marshalBuffer[sizeof(TPMS_NV_PUBLIC)];
    BYTE *buffer;
    HASH_STATE hashState;

    // Marshal public area
    buffer = marshalBuffer;
    nameSize = TPMS_NV_PUBLIC_Marshal(nvIndex->publicArea, buffer, NULL);
    // marshalbuffer

    // hash public area
    digestSize = CryptHashStart(&hashState, nvIndex->publicArea.nameAlg);
    CryptoDigestUpdate(&hashState, nameSize, marshalBuffer);

    // Complete digest leaving room for the nameAlg
    CryptoHashEnd(&hashState, digestSize, &name->buffer[2]);

    // Include the nameAlg
    UINT16_TO_BYTE_ARRAY(nvIndex->publicArea.nameAlg, name->b.buffer);
    name->t.size = digestSize + 2;
    return name;
}

8.4.5.14 NvGetNameByIndexHandle()

This function is used to compute the Name of an NV Index referenced by handle.

The name buffer receives the bytes of the Name and the return value is the number of octets in the Name.

This function requires that the NV Index is defined.

TPM2B_NAME *
NvGetNameByIndexHandle(
    TPMI_RH_NV_INDEX handle,  // IN: handle of the index
    TPM2B_NAME *name          // OUT: name of the index
)
{
    TPM2B_NAME *name;
    TArray byte[];
    UINT16 nameSize;
    BYTE marshalBuffer[sizeof(TPMS_NV_PUBLIC)];
    BYTE *buffer;
    HASH_STATE hashState;

    // Marshal public area
    buffer = marshalBuffer;
    nameSize = TPMS_NV_PUBLIC_Marshal(nvIndex->publicArea, buffer, NULL);
    // marshalbuffer

    // hash public area
    digestSize = CryptHashStart(&hashState, nvIndex->publicArea.nameAlg);
    CryptoDigestUpdate(&hashState, nameSize, marshalBuffer);

    // Complete digest leaving room for the nameAlg
    CryptoHashEnd(&hashState, digestSize, &name->b.buffer[2]);

    // Include the nameAlg
    UINT16_TO_BYTE_ARRAY(nvIndex->publicArea.nameAlg, name->b.buffer);
    name->t.size = digestSize + 2;
    return name;
8.4.5.15 NvDefineIndex()

This function is used to assign NV memory to an NV Index.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_SPACE</td>
<td>insufficient NV space</td>
</tr>
</tbody>
</table>

```c
TPM_RC
NvDefineIndex(
    TMPS_NV_PUBLIC *publicArea, // IN: A template for an area to create.
    TPM2B_AUTH *authValue     // IN: The initial authorization value
) {
    NV_INDEX nvIndex;          // the index data
    UINT16 entrySize;          // size of entry
    TPM_RC result;
    // entrySize = sizeof(NV_INDEX);
    if(!IS_ATTRIBUTE(publicArea->attributes, TPMA_NV, ORDERLY))
        entrySize += publicArea->dataSize;
    // Check if we have enough space to create the NV Index
    if(!NvTestSpace(entrySize, TRUE, IsNvCounterIndex(publicArea->attributes)))
        return TPM_RC_NV_SPACE;
    // if the index to be defined is RAM backed, check RAM space availability
    if(IS_ATTRIBUTE(publicArea->attributes, TPMA_NV, ORDERLY) && !NvRamTestSpaceIndex(publicArea->dataSize))
        return TPM_RC_NV_SPACE;
    // Copy input value to nvBuffer
    nvIndex.publicArea = *publicArea;
    // Copy the authValue
    nvIndex.authValue = *authValue;
    // Add index to NV memory
    result = NvAdd(entrySize, sizeof(NV_INDEX), TPM_RH_UNASSIGNED, (BYTE *)&nvIndex);
    if(result == TPM_RC_SUCCESS)
        // If the data of NV Index is RAM backed, add the data area in RAM as well
        if(IS_ATTRIBUTE(publicArea->attributes, TPMA_NV, ORDERLY))
            NvAddRAM(publicArea);
    return result;
```

8.4.5.16 NvAddEvictObject()

This function is used to assign NV memory to a persistent object.
<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_HANDLE</td>
<td>the requested handle is already in use</td>
</tr>
<tr>
<td>TPM_RC_NV_SPACE</td>
<td>insufficient NV space</td>
</tr>
</tbody>
</table>

877 **TPM_RC**
878 NvAddEvictObject(  
879    TPMI_DH_OBJECT evictHandle, // IN: new evict handle  
880    OBJECT *object    // IN: object to be added  
881 }
882 {  
883    TPM_HANDLE temp = object->evictHandle;  
884    TPM_RC result;  
885    // Check if we have enough space to add the evict object  
886    // An evict object needs 8 bytes in index table + sizeof OBJECT  
887    // In this implementation, the only resource limitation is the available NV  
888    // space. Other implementation may have other limitation on evict object  
889    // handle space  
890    if(!NvTestSpace(sizeof(OBJECT) + sizeof(TPM_HANDLE), FALSE, FALSE))  
891        return TPM_RC_NV_SPACE;  
892
893    // Set evict attribute and handle  
894    object->attributes.evict = SET;  
895    object->evictHandle = evictHandle;  
896
897    // Now put this in NV  
898    result = NvAdd(sizeof(OBJECT), sizeof(OBJECT), evictHandle, (BYTE *)object);  
899
900    // Put things back the way they were  
901    object->attributes.evict = CLEAR;  
902    object->evictHandle = temp;  
903
904    return result;  
905 }
906
8.4.5.17 **NvDeleteIndex()**

This function is used to delete an NV Index.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>NV is not accessible</td>
</tr>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>NV is rate limiting</td>
</tr>
</tbody>
</table>

907 **TPM_RC**
908 NvDeleteIndex(  
909    NV_INDEX *nvIndex,        // IN: an in RAM index descriptor  
910    NV_REF entityAddr        // IN: location in NV  
911 )
912 {  
913    TPM_RC result;  
914    // if(nvIndex != NULL)  
915    {  
916        // Whenever a counter is deleted, make sure that the MaxCounter value is  
917        // updated to reflect the value  
918        if(IsNvCounterIndex(nvIndex->publicArea.attributes)  
919            && IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, WRITTEN))  
920            NvUpdateMaxCount(NvGetUINT64Data(nvIndex, entityAddr));  
921        result = NvDelete(entityAddr);  
922        if(result != TPM_RC_SUCCESS)  
923            return result;  
924    }
925    return result;  
926 }
return result;

// If the NV Index is RAM backed, delete the RAM data as well
if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, ORDERLY))
NvDeleteRAM(nvIndex->publicArea.nvIndex);
NvIndexCacheInit();
}
return TPM_RC_SUCCESS;
}

8.4.5.18 NvDeleteEvict()

This function will delete a NV evict object. Will return success if object deleted or if it does not exist

TPM_RC
NvDeleteEvict(
    TPM_HANDLE       handle  // IN: handle of entity to be deleted
)
{
    NV_REF    entityAddr = NvFindEvict(handle, NULL);  // pointer to entity
    TPM_RC    result = TPM_RC_SUCCESS;
    //
    if(entityAddr != 0)
        result = NvDelete(entityAddr);
    return result;
}

8.4.5.19 NvFlushHierarchy()

This function will delete persistent objects belonging to the indicated hierarchy. If the storage hierarchy is selected, the function will also delete any NV Index defined using ownerAuth.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>NV is unavailable because of rate limit</td>
</tr>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>NV is inaccessible</td>
</tr>
</tbody>
</table>

TPM_RC
NvFlushHierarchy(
    TPMI_RH_HIERARCHY    hierarchy  // IN: hierarchy to be flushed.
)
{
    NV_REF                   iter = NV_REF_INIT;
    NV_REF                   currentAddr;
    TPM_HANDLE               entityHandle;
    TPM_RC                   result = TPM_RC_SUCCESS;
    //
    while((currentAddr = NvNext(&iter, &entityHandle)) != 0)
    {
        if(HandleGetType(entityHandle) == TPM_HT_NV_INDEX)
            {
                NV_INDEX     nvIndex;
                // If flush endorsement or platform hierarchy, no NV Index would be
                // flushed
                if(hierarchy == TPM_RH_ENDORSEMENT || hierarchy == TPM_RH_PLATFORM)
                    continue;
                // Get the index information
                NvReadNvIndexInfo(currentAddr, &nvIndex);
                // For storage hierarchy, flush OwnerCreated index
                if(!IS_ATTRIBUTE(nvIndex.publicArea.attributes, TPMA_NV,
                    PLATFORMCREATE))
970 { 
971     // Delete the index (including RAM for orderly)
972     result = NvDeleteIndex(&nvIndex, currentAddr);
973     if (result != TPM_RC_SUCCESS)
974         break;
975     // Re-iterate from beginning after a delete
976     iter = NV_REF_INIT;
977 }
978 }  
979 else if (HandleGetType(entityHandle) == TPM_HT_PERSISTENT)  
980 {  
981     OBJECT_ATTRIBUTES attributes;  
982     NvRead(&attributes,  
983             (UINT32)(currentAddr  
984                     + sizeof(TPM_HANDLE)  
985                     + offsetof(OBJECT, attributes)),  
986     sizeof(OBJECT_ATTRIBUTES));  
987     // If the evict object belongs to the hierarchy to be flushed...
988     if ((hierarchy == TPM_RH_PLATFORM && attributes.ppsHierarchy == SET)  
989         || (hierarchy == TPM_RH_OWNER && attributes.spsHierarchy == SET)  
990         || (hierarchy == TPM_RH_ENDORSEMENT  
991            && attributes.epsHierarchy == SET))
992     {  
993         // ...then delete the evict object
994         result = NvDelete(currentAddr);
995         if (result != TPM_RC_SUCCESS)
996             break;
997         // Re-iterate from beginning after a delete
998         iter = NV_REF_INIT;
999     }
1000 }  
1001 else  
1002 {  
1003     FAIL(FATAL_ERROR_INTERNAL);  
1004 }
1005 }  
1006 return result;
1007 }
1008

8.4.5.20 NvSetGlobalLock()

This function is used to SET the TPMA_NV_WRITELOCKED attribute for all NV indexes that have TPMA_NV_GLOBALLOCK SET. This function is use by TPM2_NV_GlobalWriteLock().

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>NV is unavailable because of rate limit</td>
</tr>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>NV is inaccessible</td>
</tr>
</tbody>
</table>

1009 TPM_RC
1010 NvSetGlobalLock(  
1011     void  
1012 )  
1013 {  
1014     NV_REF iter = NV_REF_INIT;  
1015     NV_RAM_REF ramIter = NV_RAM_REF_INIT;  
1016     NV_REF currentAddr;  
1017     NV_RAM_REF currentRamAddr;  
1018     TPM_RC result = TPM_RC_SUCCESS;  
1019     // Check all normal indexes  
1020     while ((currentAddr = NvNextIndex(NULL, &iter)) != 0)
1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056  
1057  
1058  
1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079  
1080  
1081  
1082  
1083  
1084  
1085  
1086  
1087  
1088  
1089  
1090  
1091  
1092  
1093  
1094  
1095  
1096  
1097  
1098  
1099  
1100  
1101  
1102  
1103  
1104  
1105  
1106  
1107  
1108  
1109  
1110  
1111  
1112  
1113  
1114  
1115  
1116  
1117  
1118  
1119  
1120  
1121  
1122  
1123  
1124  
1125  
1126  
1127  
1128  
1129  
1130  
1131  
1132  
1133  
1134  
1135  
1136  
1137  
1138  
1139  
1140  
1141  
1142  
1143  
1144  
1145  
1146  
1147  
1148  
1149  
1150  
1151  
1152  
1153  
1154  
1155  
1156  
1157  
1158  
1159  
1160  
1161  
1162  
1163  
1164  
1165  
1166  
1167  
1168  
1169  
1170  
1171  
1172  
1173  
1174  
1175  
1176  
1177  
1178  
1179  
1180  
1181  
1182  
1183  
1184  
1185  
1186  
1187  
1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  

1022  
1023  
1024  
1025  
1026  
1027  
1028  
1029  
1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039  
1040  
1041  
1042  
1043  
1044  
1045  
1046  
1047  

8.4.5.21 InsertSort()

Sort a handle into handle list in ascending order. The total handle number in the list should not exceed MAX_CAP_HANDLES.

1048  static void
1049  InsertSort(
1050  TPML_HANDLE  *handleList,  // IN/OUT: sorted handle list
1051  UINT32        count,     // IN: maximum count in the handle list
1052  TPM_HANDLE   entityHandle // IN: handle to be inserted
1053  )
1054  {
1055  UINT32          i, j;
1056  UINT32          originalCount;
1057  // For a corner case that the maximum count is 0, do nothing
1058  if(count == 0)
1059  return;
1060  // For empty list, add the handle at the beginning and return
1061  if(handleList->count == 0)
1062  {
1063      handleList->handle[0] = entityHandle;
1064      handleList->count++;
1065      return;
1066  }
1067  // Check if the maximum of the list has been reached
1068  originalCount = handleList->count;
1069  if(originalCount < count)
1070      handleList->count++;
1071  // Insert the handle to the list
1072  for(i = 0; i < originalCount; i++)
1073      for(j = handleList->count - 1; j > i; j--)
1074          handleList->handle[j] = handleList->handle[j - 1];
8.4.5.22 NvCapGetPersistent()

This function is used to get a list of handles of the persistent objects, starting at handle.

Handle must be in valid persistent object handle range, but does not have to reference an existing persistent object.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more handles available</td>
</tr>
<tr>
<td>NO</td>
<td>all the available handles has been returned</td>
</tr>
</tbody>
</table>

```c
TPMI_YES_NO NvCapGetPersistent(
    TPMI_DH_OBJECT handle,  // IN: start handle
    UINT32           count,  // IN: maximum number of returned handles
    TPM_HANDLE     *handleList,  // OUT: list of handle
)
```

```c
TPMI_YES_NO              more = NO;
NV_REF                   iter = NV_REF_INIT;
NV_REF                   currentAddr;
TPM_HANDLE               entityHandle;
```

```c
while((currentAddr = NvNextEvict(&entityHandle, &iter)) != 0)
```

```c
TPMI_YES_NO
break;
}
```

// If a slot was found, insert the handle in this position

```c
if(i < originalCount || handleList->count > originalCount)
```

```c
handleList->handle[i] = entityHandle;
return;
```
8.4.5.23 NvCapGetIndex()

This function returns a list of handles of NV indexes, starting from handle. Handle must be in the range of NV indexes, but does not have to reference an existing NV Index.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more handles to report</td>
</tr>
<tr>
<td>NO</td>
<td>all the available handles has been reported</td>
</tr>
</tbody>
</table>

```c
TPMI_YES_NO

NvCapGetIndex()

TPM_DH_OBJECT   handle, // IN: start handle
UINT32          count, // IN: max number of returned handles
TPM_HANDLE     *handleList // OUT: list of handle
{
    TPMI_YES_NO  more = NO;
    NV_REF       iter = NV_REF_INIT;
    NV_REF       currentAddr;
    TPM_HANDLE   nvHandle;
    //
pAssert(HandleGetType(handle) == TPM_HT_NV_INDEX);
    // Initialize output handle list
    handleList->count = 0;
    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES) count = MAX_CAP_HANDLES;
    while((currentAddr = NvNextIndex(&nvHandle, &iter)) != 0)
    {
        // Ignore index handles that have values less than the 'handle'
        if(nvHandle < handle)
            continue;
        // if the count of handles in the list has reached the requested count, // and there are still handles to report, set more.
        if(handleList->count == count)
            more = YES;
        // A handle with a value larger than start handle is a candidate // for return. Insert sort it to the return list. Insert sort algorithm // is chosen here for simplicity based on the assumption that the total // number of NV indexes is small. For an implementation that may allow // large number of NV indexes, a more efficient sorting algorithm may be // used here.
        InsertSort(handleList, count, nvHandle);
    }
    return more;
}
```

8.4.5.24 NvCapGetIndexNumber()

This function returns the count of NV Indexes currently defined.

```c
UINT32

NvCapGetIndexNumber(
    void
)
{
    UINT32 num = 0;
    NV_REF  iter = NV_REF_INIT;
    //
```

Family "2.0" TCG Public Review Page 309
Level 00 Revision 01.50 Copyright © TCG 2006-2019 September 18, 2018
while (NvNextIndex(NULL, &iter) != 0)
    num++;
return num;
}

8.4.5.25 NvCapGetPersistentNumber()

Function returns the count of persistent objects currently in NV memory.

UINT32
NvCapGetPersistentNumber(
    void
)
{
    UINT32 num = 0;
    NV_REF iter = NV_REF_INIT;
    TPM_HANDLE handle;
    //
    while (NvNextEvict(&handle, &iter) != 0)
        num++;
    return num;
}

8.4.5.26 NvCapGetPersistentAvail()

This function returns an estimate of the number of additional persistent objects that could be loaded into NV memory.

UINT32
NvCapGetPersistentAvail(
    void
)
{
    UINT32 availNVSpace;
    UINT32 counterNum = NvCapGetCounterNumber();
    UINT32 reserved = sizeof(NV_LIST_TERMINATOR);
    // Get the available space in NV storage
    availNVSpace = NvGetFreeBytes();
    if(counterNum < MIN_COUNTER_INDICES)
    {
        // Some space has to be reserved for counter objects.
        reserved += (MIN_COUNTER_INDICES - counterNum) * NV_INDEX_COUNTER_SIZE;
        if(reserved > availNVSpace)
            availNVSpace = 0;
        else
            availNVSpace -= reserved;
    }
    return availNVSpace / NV_EVICT_OBJECT_SIZE;
}

8.4.5.27 NvCapGetCounterNumber()

Get the number of defined NV Indexes that are counter indexes.

UINT32
NvCapGetCounterNumber(
    void
)
8.4.5.28 NvSetStartupAttributes()

Local function to set the attributes of an Index at TPM Reset and TPM Restart.

```c
static TPMA_NV
NvSetStartupAttributes(
    TPMA_NV attributes, // IN: attributes to change
    STARTUP_TYPE type    // IN: start up type
);
```

```c
{ // Clear read lock
    CLEAR_ATTRIBUTE(attributes, TPMA_NV, READLOCKED);
    // Will change a non counter index to the unwritten state if:
    // a) TPMA_NV_CLEAR_STCLEAR is SET
    // b) orderly and TPM Reset
    if(!IsNvCounterIndex(attributes))
        { if((IS_ATTRIBUTE(attributes, TPMA_NV, CLEAR_STCLEAR)
            || (IS_ATTRIBUTE(attributes, TPMA_NV, ORDERLY)
            && (type == SU_RESET)))
            CLEAR_ATTRIBUTE(attributes, TPMA_NV, WRITTEN);
        }
    // Unlock any index that is not written or that does not have
    // TPMA_NV_WRITEDEFINE SET.
    if(!IS_ATTRIBUTE(attributes, TPMA_NV, WRITTEN)
        || !IS_ATTRIBUTE(attributes, TPMA_NV, WRITEDEFINE))
        CLEAR_ATTRIBUTE(attributes, TPMA_NV, WRITELOCKED);
    return attributes;
}
```

8.4.5.29 NvEntityStartup()

This function is called at TPM_Startup(). If the startup completes a TPM Resume cycle, no action is taken. If the startup is a TPM Reset or a TPM Restart, then this function will:

a) clear read/write lock;

b) reset NV Index data that has TPMA_NV_CLEAR_STCLEAR SET; and

c) set the lower bits in orderly counters to 1 for a non-orderly startup

It is a prerequisite that NV be available for writing before this function is called.

```c
void
NvEntityStartup(
    STARTUP_TYPE type    // IN: start up type
);
```

```c
{ NV_REF iter = NV_REF_INIT;
    NV_RAM_REF ramIter = NV_RAM_REF_INIT;
```
1265  NV_REF currentAddr;  // offset points to the current entity
1266  NV_RAM_REF currentRamAddr;
1267  TPM_HANDLE nvHandle;
1268  TPM_NV attributes;
1269  //
1270  // Restore RAM index data
1271  NvRead(s_indexOrderlyRam, NV_INDEX_RAM_DATA, sizeof(s_indexOrderlyRam));
1272  // Initialize the max NV counter value
1273  NvSetMaxCount(NvGetMaxCount());
1274  // If recovering from state save, do nothing else
1275  if(type == SU_RESUME)
1276      return;
1277  // Iterate all the NV Index to clear the locks
1278  while((currentAddr = NvNextIndex(&nvHandle, &iter)) != 0)
1279  {
1280      attributes = NvReadNvIndexAttributes(currentAddr);
1281      // If this is an orderly index, defer processing until loop below
1282      if(IS_ATTRIBUTE(attributes, TPM_NV, ORDERLY))
1283          continue;
1284      // Set the attributes appropriate for this startup type
1285      attributes = NvSetStartupAttributes(attributes, type);
1286      NvWriteNvIndexAttributes(currentAddr, attributes);
1287  }
1288  // Iterate all the orderly indexes to clear the locks and initialize counters
1289  while((currentRamAddr = NvRamNext(&ramIter, NULL)) != 0)
1290  {
1291      attributes = NvReadRamIndexAttributes(currentRamAddr);
1292      attributes = NvSetStartupAttributes(attributes, type);
1293      // update attributes in RAM
1294      NvWriteRamIndexAttributes(currentRamAddr, attributes);
1295      // Set the lower bits in an orderly counter to 1 for a non-orderly startup
1296      if(IsNvCounterIndex(attributes) && (g_prevOrderlyState == SU_NONE_VALUE))
1297          {
1298              UINT64 counter;
1299              // Read the counter value last saved to NV.
1300              counter = BYTE_ARRAY_TO_UINT64(currentRamAddr + sizeof(NV_RAM_HEADER));
1301              // Set the lower bits of counter to 1's
1302              counter |= MAX_ORDERLY_COUNT;
1303              // Write back to RAM
1304              // NOTE: Do not want to force a write to NV here. The counter value will
1305              // stay in RAM until the next shutdown or rollover.
1306              UINT64_TO_BYTE_ARRAY(counter, currentRamAddr + sizeof(NV_RAM_HEADER));
1307          }
1308      return;
1309  }

8.4.5.30 NvCapGetCounterAvail()

This function returns an estimate of the number of additional counter type NV indexes that can be defined.

1321  UINT32
1322  NvCapGetCounterAvail(
void
{
    UINT32      availNVSpace;
    UINT32      availRAMSpace;
    UINT32      persistentNum = NvCapGetPersistentNumber();
    UINT32      reserved = sizeof(NV_LIST_TERMINATOR);
    //
    // Get the available space in NV storage
    availNVSpace = NvGetFreeBytes();
    //
    if(persistentNum < MIN_EVICT_OBJECTS)
    {
        // Some space has to be reserved for evict object. Adjust availNVSpace.
        reserved += (MIN_EVICT_OBJECTS - persistentNum) * NV_EVICT_OBJECT_SIZE;
        if(reserved > availNVSpace)
            availNVSpace = 0;
        else
            availNVSpace -= reserved;
    }
    // Compute the available space in RAM
    availRAMSpace = RAM_ORDERLY_END - NvRamGetEnd();
    // Return the min of counter number in NV and in RAM
    if(availNVSpace / NV_INDEX_COUNTER_SIZE > availRAMSpace / NV_RAM_INDEX_COUNTER_SIZE)
        return availRAMSpace / NV_RAM_INDEX_COUNTER_SIZE;
    else
        return availNVSpace / NV_INDEX_COUNTER_SIZE;
}

8.4.5.31 NvFindHandle()

this function returns the offset in NV memory of the entity associated with the input handle. A value of zero indicates that handle does not exist reference an existing persistent object or defined NV Index.

NV_REF
NvFindHandle(TPM_HANDLE handle)
{
    NV_REF           addr;
    NV_REF           iter = NV_REF_INIT;
    TPM_HANDLE       nextHandle;
    //
    while((addr = NvNext(&iter, &nextHandle)) != 0)
    {
        if(nextHandle == handle)
            break;
    }
    return addr;
}

8.4.6 NV Max Counter

8.4.6.1 Introduction

The TPM keeps track of the highest value of a deleted counter index. When an index is deleted, this value is updated if the deleted counter index is greater than the previous value. When a new index is created and first incremented, it will get a value that is at least one greater than any other index than any previously deleted index. This insures that it is not possible to roll back an index.
The highest counter value is kept in NV in a special end-of-list marker. This marker is only updated when an index is deleted. Otherwise, it just moves.

When the TPM starts up, it searches NV for the end of list marker and initializes an in-memory value (s_maxCounter).

8.4.6.2 NvReadMaxCount()

This function returns the max NV counter value.

```c
UINT64
NvReadMaxCount(
    void
) {
    return s_maxCounter;
}
```

8.4.6.3 NvUpdateMaxCount()

This function updates the max counter value to NV memory. This is just staging for the actual write that will occur when the NV index memory is modified.

```c
void
NvUpdateMaxCount(
    UINT64           count
) {
    if(count > s_maxCounter)
        s_maxCounter = count;
}
```

8.4.6.4 NvSetMaxCount()

This function is used at NV initialization time to set the initial value of the maximum counter.

```c
void
NvSetMaxCount(
    UINT64          value
) {
    s_maxCounter = value;
}
```

8.4.6.5 NvGetMaxCount()

Function to get the NV max counter value from the end-of-list marker

```c
UINT64
NvGetMaxCount(
    void
) {
    NV_REF               iter = NV_REF_INIT;
    NV_REF               currentAddr;
    UINT64               maxCount;
    // Find the end of list marker and initialize the NV Max Counter value.
    while((currentAddr = NvNext(&iter, NULL )) != 0);
```
1402  // 'iter' should be pointing at the end of list marker so read in the current
1403  // value of the s_maxCounter.
1404  NvRead(&maxCount, iter + sizeof(UINT32), sizeof(maxCount));
1405
1406  return maxCount;
1407  }
8.5 NvReserved.c

8.5.1 Introduction

The NV memory is divided into two areas: dynamic space for user defined NV Indices and evict objects, and reserved space for TPM persistent and state save data.

The entries in dynamic space are a linked list of entries. Each entry has, as its first field, a size. If the size field is zero, it marks the end of the list.

An allocation of an Index or evict object may use almost all of the remaining NV space such that the size field will not fit. The functions that search the list are aware of this and will terminate the search if they either find a zero size or recognize that there is insufficient space for the size field.

An Index allocation will contain an NV_INDEX structure. If the Index does not have the orderly attribute, the NV_INDEX is followed immediately by the NV data.

An evict object entry contains a handle followed by an OBJECT structure. This results in both the Index and Evict Object having an identifying handle as the first field following the size field.

When an Index has the orderly attribute, the data is kept in RAM. This RAM is saved to backing store in NV memory on any orderly shutdown. The entries in orderly memory are also a linked list using a size field as the first entry. As with the NV memory, the list is terminated by a zero size field or when the last entry leaves insufficient space for the terminating size field.

The attributes of an orderly index are maintained in RAM memory in order to reduce the number of NV writes needed for orderly data. When an orderly index is created, an entry is made in the dynamic NV memory space that holds the Index authorizations (authPolicy and authValue) and the size of the data. This entry is only modified if the authValue of the index is changed. The more volatile data of the index is kept in RAM. When an orderly Index is created or deleted, the RAM data is copied to NV backing store so that the image in the backing store matches the layout of RAM. In normal operation. The RAM data is also copied on any orderly shutdown. In normal operation, the only other reason for writing to the backing store for RAM is when a counter is first written (TPMA_NV_WRITTEN changes from CLEAR to SET) or when a counter "rolls over."

Static space contains items that are individually modifiable. The values are in the gp PERSISTEND_DATA structure in RAM and mapped to locations in NV.

8.5.2 Includes, Defines

```c
#define NV_C
#include "Tpm.h"
```

8.5.3 Functions

8.5.3.1 NvInitStatic()

This function initializes the static variables used in the NV subsystem.

```c
static void
NvInitStatic(
void
)
{
    // In some implementations, the end of NV is variable and is set at boot time.
    // This value will be the same for each boot, but is not necessarily known
    // at compile time.
    s_evictNvEnd = (NV_REF)NV_MEMORY_SIZE;
    return;
```
8.5.3.2 NvCheckState()

Function to check the NV state by accessing the platform-specific function to get the NV state. The result state is registered in s_NvIsAvailable that will be reported by NvIsAvailable().

This function is called at the beginning of ExecuteCommand() before any potential check of g_NvStatus.

```c
void NvCheckState()
{
    int func_return;
    //
    func_return = _plat__IsNvAvailable();
    if(func_return == 0)
        g_NvStatus = TPM_RC_SUCCESS;
    else if(func_return == 1)
        g_NvStatus = TPM_RC_NV_UNAVAILABLE;
    else
        g_NvStatus = TPM_RC_NV_RATE;
    return;
}
```

8.5.3.3 NvCommit

This is a wrapper for the platform function to commit pending NV writes.

```c
BOOL NvCommit()
{
    return (_plat__NvCommit() == 0);
}
```

8.5.3.4 NvPowerOn()

This function is called at _TPM_Init() to initialize the NV environment.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>all NV was initialized</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>the NV containing saved state had an error and TPM2_Startup(CLEAR) is required</td>
</tr>
</tbody>
</table>

```c
BOOL NvPowerOn()
{
    int nvError = 0;
    // If power was lost, need to re-establish the RAM data that is loaded from
    // NV and initialize the static variables
    if(g_powerWasLost)
    {
        if((nvError = _plat__NVEnable(0)) < 0)
            FAIL(FATAL_ERROR_NV_UNRECOVERABLE);
    }
```
8.5.3.5  NvManufacture()

This function initializes the NV system at pre-install time.

This function should only be called in a manufacturing environment or in a simulation.

The layout of NV memory space is an implementation choice.

```c
void
NvManufacture(
    void
)
{
    #if SIMULATION
        // Simulate the NV memory being in the erased state.
        _plat__NvMemoryClear(0, NV_MEMORY_SIZE);
    #endif
    // Initialize static variables
    NvInitStatic();
    // Clear the RAM used for Orderly Index data
    MemorySet(s_indexOrderlyRam, 0, RAM_INDEX_SPACE);
    // Write that Orderly Index data to NV
    NvUpdateIndexOrderlyData();
    // Initialize the next offset of the first entry in evict/index list to 0 (the end of list marker) and the initial s_maxCounterValue;
    NvSetMaxCounterValue(0);
    // Put the end of list marker at the end of memory. This contains the MaxCount value as well as the end marker.
    NvWriteNvListEnd(NV_USER_DYNAMIC);
    return;
}
```

8.5.3.6  NvRead()

This function is used to move reserved data from NV memory to RAM.

```c
void
NvRead(
    UINT32           *outBuffer,  // OUT: buffer to receive data
    UINT32           nvOffset,   // IN: offset in NV of value
    UINT32           size       // IN: size of the value to read
)
{
    // Input type should be valid
    pAssert(nvOffset + size < NV_MEMORY_SIZE);
    _plat__NvMemoryRead(nvOffset, size, outBuffer);
    return;
}
```

8.5.3.7  NvWrite()

This function is used to post reserved data for writing to NV memory. Before the TPM completes the operation, the value will be written.

```c
void
NvWrite(
    UINT32           nvOffset,   // IN: location in NV to receive data
)
```
8.5.3.8 NvUpdatePersistent()

This function is used to update a value in the PERSISTENT_DATA structure and commits the value to NV.

```c
void NvUpdatePersistent(
    UINT32 offset, // IN: location in PERMANENT_DATA to be updated
    UINT32 size,   // IN: size of the value
    void *buffer   // IN: the new data
)
{
    pAssert(offset + size <= sizeof(gp));
    MemoryCopy(&gp + offset, buffer, size);
    NvWrite(offset, size, buffer);
}
```

8.5.3.9 NvClearPersistent()

This function is used to clear a persistent data entry and commit it to NV

```c
void NvClearPersistent(
    UINT32 offset, // IN: the offset in the PERMANENT_DATA structure to be cleared (zeroed)
    UINT32 size    // IN: number of bytes to clear
)
{
    pAssert(offset + size <= sizeof(gp));
    MemorySet((&gp) + offset, 0, size);
    NvWrite(offset, size, (&gp) + offset);
}
```

8.5.3.10 NvReadPersistent()

This function reads persistent data to the RAM copy of the gp structure.

```c
void NvReadPersistent(
    void
)
{
    NvRead(&gp, NV_PERSISTENT_DATA, sizeof(gp));
    return;
}
```
8.6 Object.c

8.6.1 Introduction

This file contains the functions that manage the object store of the TPM.

8.6.2 Includes and Data Definitions

```c
#define OBJECT_C
#include "Tpm.h"
```

8.6.3 Functions

8.6.3.1 ObjectFlush()

This function marks an object slot as available. Since there is no checking of the input parameters, it should be used judiciously.

NOTE: This could be converted to a macro.

```c
void ObjectFlush( OBJECT *object )
{
    object->attributes.occupied = CLEAR;
}
```

8.6.3.2 ObjectSetInUse()

This access function sets the occupied attribute of an object slot.

```c
void ObjectSetInUse( OBJECT *object )
{
    object->attributes.occupied = SET;
}
```

8.6.3.3 ObjectStartup()

This function is called at TPM2_Startup() to initialize the object subsystem.

```c
void ObjectStartup( void )
{
    UINT32 i;
    // object slots initialization
    for(i = 0; i < MAX_LOADED_OBJECTS; i++)
    {
        // Set the slot to not occupied
        ObjectFlush(&s_objects[i]);
    }
```
8.6.3.4 ObjectCleanupEvict()

In this implementation, a persistent object is moved from NV into an object slot for processing. It is flushed after command execution. This function is called from ExecuteCommand().

```c
void ObjectCleanupEvict()
{
    UINT32 i;
    // This has to be iterated because a command may have two handles
    // and they may both be persistent.
    // This could be made to be more efficient so that a search is not needed.
    for(i = 0; i < MAX_LOADED_OBJECTS; i++)
    {
        // If an object is a temporary evict object, flush it from slot
        OBJECT *object = &s_objects[i];
        if(object->attributes.evict == SET)
            ObjectFlush(object);
    }
    return;
}
```

8.6.3.5 IsObjectPresent()

This function checks to see if a transient handle references a loaded object. This routine should not be called if the handle is not a transient handle. The function validates that the handle is in the implementation-dependent allowed in range for loaded transient objects.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>handle references a loaded object</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>handle is not an object handle, or it does not reference to a loaded object</td>
</tr>
</tbody>
</table>

```c
BOOL IsObjectPresent(TPMI_DH_OBJECT handle) // IN: handle to be checked
{
    UINT32 slotIndex = handle - TRANSIENT_FIRST;
    // Since the handle is just an index into the array that is zero based, any
    // handle value outside of the range of:
    // TRANSIENT_FIRST -- (TRANSIENT_FIRST + MAX_LOADED_OBJECTS - 1)
    // will now be greater than or equal to MAX_LOADED_OBJECTS
    if(slotIndex >= MAX_LOADED_OBJECTS)
        return FALSE;
    // Indicate if the slot is occupied
    return (s_objects[slotIndex].attributes.occupied == TRUE);
}
```

8.6.3.6 ObjectIsSequence()

This function is used to check if the object is a sequence object. This function should not be called if the handle does not reference a loaded object.
**Return Value** | **Meaning**  
--- | ---  
TRUE(1) | object is an HMAC, hash, or event sequence object  
FALSE(0) | object is not an HMAC, hash, or event sequence object  

```c
BOOL ObjectIsSequence(
    OBJECT *object    // IN: handle to be checked
) {
    pAssert(object != NULL);
    return (object->attributes.hmacSeq == SET
        || object->attributes.hashSeq == SET
        || object->attributes.eventSeq == SET);
}
```

### 8.6.3.7 HandleToObject()

This function is used to find the object structure associated with a handle.

This function requires that handle references a loaded object or a permanent handle.

```c
OBJECT* HandleToObject(
    TPMI_DH_OBJECT handle    // IN: handle of the object
) {
    UINT32 index;
    // Return NULL if the handle references a permanent handle because there is no
    // associated OBJECT.
    if (HandleGetType(handle) == TPM_HT_PERMANENT)
        return NULL;
    // In this implementation, the handle is determined by the slot occupied by the
    // object.
    index = handle - TRANSIENT_FIRST;
    pAssert(index < MAX_LOADED_OBJECTS);
    pAssert(s_objects[index].attributes.occupied);
    return &s_objects[index];
}
```

### 8.6.3.8 ObjectGetNameAlg()

This function is used to get the Name algorithm of a object.

This function requires that object references a loaded object.

```c
TPMI_ALG_HASH ObjectGetNameAlg(
    OBJECT *object    // IN: handle of the object
) {
    return object->publicArea.nameAlg;
}
```

### 8.6.3.9 GetQualifiedName()

This function returns the Qualified Name of the object. In this implementation, the Qualified Name is computed when the object is loaded and is saved in the internal representation of the object. The
alternative would be to retain the Name of the parent and compute the QN when needed. This would take the same amount of space so it is not recommended that the alternate be used.

This function requires that handle references a loaded object.

```c
void
GetQualifiedName(
    TPMI_DH_OBJECT       handle,   // IN: handle of the object
    TPM2B_NAME          *qualifiedName // OUT: qualified name of the object
)
{
    OBJECT   *object;
    //
    switch(HandleGetType(handle)) {
    case TPM_HT_PERMANENT:
        qualifiedName->t.size = sizeof(TPM_HANDLE);
        UINT32_TO_BYTE_ARRAY(handle, qualifiedName->t.name);
        break;
    case TPM_HT_TRANSIENT:
        object = HandleToObject(handle);
        if(object == NULL || object->publicArea.nameAlg == TPM_ALG_NULL)
            qualifiedName->t.size = 0;
        else
            // Copy the name
            *qualifiedName = object->qualifiedName;
        break;
    default:
        FAIL(FATAL_ERROR_INTERNAL);
    }
    return;
}
```

**8.6.3.10 ObjectGetHierarchy()**

This function returns the handle for the hierarchy of an object.

```c
TPMI_RH_HIERARCHY
ObjectGetHierarchy(
    OBJECT   *object     // IN : object
)
{
    if(object->attributes.spsHierarchy)
    {
        return TPM_RH_OWNER;
    }
    else if(object->attributes.epsHierarchy)
    {
        return TPM_RH_ENDORSEMENT;
    }
    else if(object->attributes.ppsHierarchy)
    {
        return TPM_RH_PLATFORM;
    }
    else
    {
        return TPM_RH_NULL;
    }
}
8.6.3.11 GetHierarchy()

This function returns the handle of the hierarchy to which a handle belongs. This function is similar to ObjectGetHierarchy() but this routine takes a handle but ObjectGetHierarchy() takes an pointer to an object.

This function requires that handle references a loaded object.

```c
TPMI_RH_HIERARCHY GetHierarchy(
    TPMI_DH_OBJECT handle   // IN : object handle
) {
    OBJECT *object = HandleToObject(handle);
    // return ObjectGetHierarchy(object);
}
```

8.6.3.12 FindEmptyObjectSlot()

This function finds an open object slot, if any. It will clear the attributes but will not set the occupied attribute. This is so that a slot may be used and discarded if everything does not go as planned.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>no open slot found</td>
</tr>
<tr>
<td>!= NULL</td>
<td>pointer to available slot</td>
</tr>
</tbody>
</table>

```c
OBJECT *
FindEmptyObjectSlot(
    TPMI_DH_OBJECT *handle   // OUT: (optional)
) {
    UINT32 i;
    OBJECT *object;
    // for(i = 0; i < MAX_LOADED_OBJECTS; i++)
    { object = &s_objects[i];
      if(object->attributes.occupied == CLEAR)
      { if(handle)
        *handle = i + TRANSIENT_FIRST;
        // Initialize the object attributes
        MemorySet(&object->attributes, 0, sizeof(OBJECT_ATTRIBUTES));
        return object;
      }
    } return NULL;
}
```

8.6.3.13 ObjectAllocateSlot()

This function is used to allocate a slot in internal object array.

```c
OBJECT *
ObjectAllocateSlot(
    TPMI_DH_OBJECT *handle   // OUT: handle of allocated object
) {
    OBJECT *object = FindEmptyObjectSlot(handle);
```
187 //
188 if(object != NULL)
189 {
190     // if found, mark as occupied
191     ObjectSetInUse(object);
192 }
193 return object;
194 }

8.6.3.14 ObjectSetLoadedAttributes()

This function sets the internal attributes for a loaded object. It is called to finalize the OBJECT attributes (not the TPMA_OBJECT attributes) for a loaded object.

195 void
196 ObjectSetLoadedAttributes(
197     OBJECT *object, // IN: object attributes to finalize
198     TPM_HANDLE parentHandle // IN: the parent handle
199 )
200 {
201     OBJECT *parent = HandleToObject(parentHandle);
202     TPMA_OBJECT objectAttributes = object->publicArea.objectAttributes;
203     // Copy the stClear attribute from the public area. This could be overwritten
204     // if the parent has stClear SET
205     object->attributes.stClear =
206         IS_ATTRIBUTE(objectAttributes, TPMA_OBJECT, stClear);
207     // If parent handle is a permanent handle, it is a primary (unless it is NULL
208     if(parent == NULL)
209        {
210             object->attributes.primary = SET;
211             switch(parentHandle)
212             {
213                 case TPM_RH_ENDORSEMENT:
214                     object->attributes.epsHierarchy = SET;
215                     break;
216                 case TPM_RH_OWNER:
217                     object->attributes.spsHierarchy = SET;
218                     break;
219                 case TPM_RH_PLATFORM:
220                     object->attributes.ppsHierarchy = SET;
221                     break;
222                 default:
223                     // Treat the temporary attribute as a hierarchy
224                     object->attributes.temporary = SET;
225                     object->attributes.primary = CLEAR;
226                     break;
227             }
228         }
229     else
230         {
231             // is this a stClear object
232             object->attributes.stClear =
233                 (IS_ATTRIBUTE(objectAttributes, TPMA_OBJECT, stClear)
234                 || (parent->attributes.stClear == SET));
235             object->attributes.epsHierarchy = parent->attributes.epsHierarchy;
236             object->attributes.spsHierarchy = parent->attributes.spsHierarchy;
237             object->attributes.ppsHierarchy = parent->attributes.ppsHierarchy;
238             // An object is temporary if its parent is temporary or if the object
239             // is external
240             object->attributes.temporary = parent->attributes.temporary
241                 || object->attributes.external;
242         }
243     // If this is an external object, set the QN == name but don't SET other...
245  // key properties ('parent' or 'derived')
246  if(object->attributes.external)
247     object->qualifiedName = object->name;
248  else
249     {
250      // check attributes for different types of parents
251      if(IS_ATTRIBUTE(objectAttributes, TPMA_OBJECT, restricted)
252         && !object->attributes.publicOnly
253         && IS_ATTRIBUTE(objectAttributes, TPMA_OBJECT, decrypt)
254         && object->publicArea.nameAlg != TPM_ALG_NULL)
255         {
256          // This is a parent. If it is not a KEYEDHASH, it is an ordinary parent.
257          // Otherwise, it is a derivation parent.
258          if(object->publicArea.type == TPM_ALG_KEYEDHASH)
259             object->attributes.derivation = SET;
260          else
261             object->attributes.isParent = SET;
262         }
263      ComputeQualifiedName(parentHandle, object->publicArea.nameAlg,
264                                &object->name, &object->qualifiedName);
265     }
266  // Set slot occupied
267  ObjectSetInUse(object);
268  return;
269 }

8.6.3.15  ObjectLoad()

Common function to load an object. A loaded object has its public area validated (unless its nameAlg is
TPM_ALG_NULL). If a sensitive part is loaded, it is verified to be correct and if both public and sensitive
parts are loaded, then the cryptographic binding between the objects is validated. This function does not
cause the allocated slot to be marked as in use.

270  TPM_RC
271  ObjectLoad(
272      OBJECT *object,     // IN: pointer to object slot
273      // IN: object
274      OBJECT *parent,     // IN: (optional) the parent object
275      TPM_PUBLIC *publicArea,  // IN: public area to be installed in the object
276      TPM_SENSITIVE *sensitive,   // IN: (optional) sensitive area to be
277      // installed in the object
278      TPM_RC blamePublic,    // IN: parameter number to associate with the
279      // publicArea errors
280      TPM_RC blameSensitive,   // IN: parameter number to associate with the
281      // sensitive area errors
282      TPM2B_NAME *name       // IN: (optional)
283 )
284 {
285   TPM_RC result = TPM_RC_SUCCESS;
286   BOOL doCheck;
287  //
288  // Do validations of public area object descriptions
289  if(sensitive == NULL || publicArea->nameAlg == TPM_ALG_NULL)
290     {
291      // Need to have schemes checked so that we do the right thing with the
292      // public key.
293      result = SchemeChecks(NULL, publicArea);
294     }  
295  else
296     {
297      // For any sensitive area, make sure that the seedSize is no larger than the
298      // digest size of nameAlg
299  //}
if(sensitive->seedValue.t.size > CryptHashGetDigestSize(publicArea->nameAlg))
    return TPM_RCS_KEY_SIZE + blameSensitive;
// Check attributes and schemes for consistency
result = PublicAttributesValidation(parent, publicArea);
}
if(result != TPM_RC_SUCCESS)
    return RcSafeAddToResult(result, blamePublic);
// If object == NULL, then this is an import. For import, load is not called
// unless the parent is fixedTPM.
if(object == NULL)
    doCheck = TRUE;  //
// If the parent is not NULL, then this is an ordinary load and we only check
// if the parent is not fixedTPM
else if(parent != NULL)
    doCheck = !IS_ATTRIBUTE(parent->publicArea.objectAttributes,
        TPMA_OBJECT, fixedTPM);
else
    // This is a loadExternal. Check everything.
    // Note: the check functions will filter things based on the name algorithm
    // and whether or not both parts are loaded.
    doCheck = TRUE;
    // Note: the parent will be NULL if this is a load external. CryptValidateKeys()
    // will only check the parts that need to be checked based on the settings
    // of publicOnly and nameAlg.
    // Note: For an RSA key, the keys sizes are checked but the binding is not
    // checked.
    if(doCheck)
    {
        // Do the cryptographic key validation
        result = CryptValidateKeys(publicArea, sensitive, blamePublic,
            blameSensitive);
    }
    // If this is an import, we are done
    if(object == NULL || result != TPM_RC_SUCCESS)
        return result;
    // Set the name, if one was provided
    if(name != NULL)
        object->name = *name;
    else
        object->name.t.size = 0;
    // Initialize public
    object->publicArea = *publicArea;
    // If there is a sensitive area, load it
    if(sensitive == NULL)
        object->attributes.publicOnly = SET;
    else
    {
        // If this is an RSA key that is not a parent, complete the load by
        // computing the private exponent.
        if(publicArea->type == ALG_RSA_VALUE)
            result = CryptRsaloadPrivateExponent(object);
    }
    return result;
}

8.6.3.16 AllocateSequenceSlot()

This function allocates a sequence slot and initializes the parts that are used by the normal objects so that a sequence object is not inadvertently used for an operation that is not appropriate for a sequence.
### AllocateSequenceSlot()

```c
static HASH_OBJECT *
AllocateSequenceSlot(
    TPM_HANDLE      *newHandle,  // OUT: receives the allocated handle
    TPM2B_AUTH      *auth       // IN: the authValue for the slot
)
{
    HASH_OBJECT      *object = (HASH_OBJECT *)ObjectAllocateSlot(newHandle);
    // Validate that the proper location of the hash state data relative to the
    // object state data. It would be good if this could have been done at compile
    // time but it can't so do it in something that can be removed after debug.
    cAssert(offsetof(HASH_OBJECT, auth) == offsetof(OBJECT, publicArea.authPolicy));

    if(object != NULL)
    {
        // Set the common values that a sequence object shares with an ordinary object
        // First, clear all attributes
        MemorySet(&object->objectAttributes, 0, sizeof(TPMA_OBJECT));

        // The type is TPM_ALG_NULL
        object->type = TPM_ALG_NULL;

        // This has no name algorithm and the name is the Empty Buffer
        object->nameAlg = TPM_ALG_NULL;

        // A sequence object is considered to be in the NULL hierarchy so it should
        // be marked as temporary so that it can't be persisted
        object->attributes.temporary = SET;

        // A sequence object is DA exempt.
        SET_ATTRIBUTE(object->objectAttributes, TPMA_OBJECT, noDA);

        // Copy the authorization value
        if(auth != NULL)
            object->auth = *auth;
        else
            object->auth.t.size = 0;
        return object;
    }
}
```

8.6.3.17 **ObjectCreateHMACSequence()**

This function creates an internal HMAC sequence object.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>if there is no free slot for an object</td>
</tr>
</tbody>
</table>

```c
TPM_RC
ObjectCreateHMACSequence(
    TPMI_ALG_HASH    hashAlg,  // IN: hash algorithm
    OBJECT           *keyObject, // IN: the object containing the HMAC key
    TPM2B_AUTH       *auth,     // IN: authValue
    TPMI_DH_OBJECT   *newHandle // OUT: HMAC sequence object handle
)
{
    HASH_OBJECT      *hmacObject;
    // Try to allocate a slot for new object
    hmacObject = AllocateSequenceSlot(newHandle, auth);
```
8.6.3.18 ObjectCreateHashSequence()

This function creates a hash sequence object.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>if there is no free slot for an object</td>
</tr>
</tbody>
</table>

8.6.3.19 ObjectCreateEventSequence()

This function creates an event sequence object.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>if there is no free slot for an object</td>
</tr>
</tbody>
</table>
HASH_OBJECT *hashObject = AllocateSequenceSlot(newHandle, auth);

UINT32 count;

TPM_ALG_ID hash;

// See if slot allocated
if(hashObject == NULL)
  return TPM_RC_OBJECT_MEMORY;

// Set the event sequence attribute
hashObject->attributes.eventSeq = SET;

// Initialize hash states for each implemented PCR algorithms
for(count = 0; (hash = CryptHashGetAlgByIndex(count)) != TPM_ALG_NULL; count++)
  CryptHashStart(&hashObject->state.hashState[count], hash);

return TPM_RC_SUCCESS;

8.6.3.20 ObjectTerminateEvent()

This function is called to close out the event sequence and clean up the hash context states.

```
void ObjectTerminateEvent()
{
  HASH_OBJECT *hashObject;
  int count;
  BYTE buffer[MAX_DIGEST_SIZE];

  // hashObject = (HASH_OBJECT *)HandleToObject(g_DRTMHandle);
  // Don't assume that this is a proper sequence object
  if(hashObject->attributes.eventSeq)
  {
    // If it is, close any open hash contexts. This is done in case
    // the cryptographic implementation has some context values that need to be
    // cleaned up (hygiene).
    //
    for(count = 0; CryptHashGetAlgByIndex(count) != TPM_ALG_NULL; count++)
      { CryptHashEnd(&hashObject->state.hashState[count], 0, buffer); }
    // Flush sequence object
    FlushObject(g_DRTMHandle);
    g_DRTMHandle = TPM_RH_UNASSIGNED;
  }
}
```

8.6.3.21 ObjectContextLoad()

This function loads an object from a saved object context.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>if there is no free slot for an object</td>
</tr>
<tr>
<td>!= NULL</td>
<td>points to the loaded object</td>
</tr>
</tbody>
</table>

```
OBJECT * ObjectContextLoad(
    ANY_OBJECT_BUFFER *object, // IN: pointer to object structure in saved
    TPMI_DH_OBJECT *handle     // OUT: object handle
) {
```
OBJECT *newObject = ObjectAllocateSlot(handle);

// Try to allocate a slot for new object
if (newObject != NULL) {
    // Copy the first part of the object
    MemoryCopy(newObject, object, offsetof(HASH_OBJECT, state));
    // See if this is a sequence object
    if (ObjectIsSequence(newObject)) {
        // If this is a sequence object, import the data
        SequenceDataImport((HASH_OBJECT *)newObject,
                            (HASH_OBJECT_BUFFER *)object);
    }
    else {
        // Copy input object data to internal structure
        MemoryCopy(newObject, object, sizeof(OBJECT));
    }
}
return newObject;

8.6.3.22 FlushObject()

This function frees an object slot.
This function requires that the object is loaded.

void FlushObject(
    TPMI_DH_OBJECT handle // IN: handle to be freed
) {
    UINT32 index = handle - TRANSIENT_FIRST;
    // Clear all the object attributes
    MemorySet((BYTE*)&(s_objects[index].attributes), 0, sizeof(OBJECT_ATTRIBUTES));
    return;
}

8.6.3.23 ObjectFlushHierarchy()

This function is called to flush all the loaded transient objects associated with a hierarchy when the hierarchy is disabled.

void ObjectFlushHierarchy(
    TPMI_RH_HIERARCHY hierarchy // IN: hierarchy to be flush
) {
    UINT16 i;
    // iterate object slots
    for (i = 0; i < MAX_LOADED_OBJECTS; i++) {
        if (s_objects[i].attributes.occupied) {
            switch (hierarchy)
8.6.3.24 ObjectLoadEvict()

This function loads a persistent object into a transient object slot.

This function requires that \textit{handle} is associated with a persistent object.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HANDLE</td>
<td>the persistent object does not exist or the associated hierarchy is disabled.</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>no object slot</td>
</tr>
</tbody>
</table>

```
578  TPM_RC
579  ObjectLoadEvict(
580      TPM_HANDLE      *handle,  // IN:OUT: evict object handle. If success, it
581          COMMAND_INDEX    commandIndex  // IN: the command being processed
582  )
583  {  
584      TPM_RC          result;
585      TPM_HANDLE      evictHandle = *handle;  // Save the evict handle
586      OBJECT          *object;
587      // If this is an index that references a persistent object created by
588      // the platform, then return TPM_RC_HANDLE if the phEnable is FALSE
589      if(*handle >= PLATFORM_PERSISTENT)
590      {  
591          // belongs to platform
592          if(g_phEnable == CLEAR)
593              return TPM_RC_HANDLE;
594      }
595      // belongs to owner
596      else if(gc.shEnable == CLEAR)
597          return TPM_RC_HANDLE;
598      // Try to allocate a slot for an object
599      object = ObjectAllocateSlot(handle);
600      if(object == NULL)
601          return TPM_RC_OBJECT_MEMORY;
602      // Copy persistent object to transient object slot. A TPM_RC_HANDLE
603      // may be returned at this point. This will mark the slot as containing
604      // a transient object so that it will be flushed at the end of the
607     // command
608     result = NvGetEvictObject(evictHandle, object);
609     // Bail out if this failed
610     if (result != TPM_RC_SUCCESS)
611         return result;
612     // check the object to see if it is in the endorsement hierarchy
613     // if it is and this is not a TPM2_EvictControl() command, indicate
614     // that the hierarchy is disabled.
615     // If the associated hierarchy is disabled, make it look like the
616     // handle is not defined
617     if (ObjectGetHierarchy(object) == TPM_RH_ENDORSEMENT
618         && gc.ehEnable == CLEAR
619         && GetCommandCode(commandIndex) != TPM_CC_EvictControl)
620         return TPM_RC_HANDLE;
621     }
622 }
623
624 8.6.3.25 ObjectComputeName()

This does the name computation from a public area (can be marshaled or not).

625  TPM2B_NAME *
626  ObjectComputeName(  
627     UINT32 size,       // IN: the size of the area to digest
628     BYTE *publicArea, // IN: the public area to digest
629     TPM_ALG_ID nameAlg, // IN: the hash algorithm to use
630     TPM2B_NAME *name   // OUT: Computed name
631 )
632 {
633     // Hash the publicArea into the name buffer leaving room for the nameAlg
634     name->t.size = CryptHashBlock(nameAlg, size, publicArea,
635         sizeof(name->t.name) - 2,
636         &name->t.name[2]);
637     // set the nameAlg
638     UINT16_TO_BYTE_ARRAY(nameAlg, name->t.name);
639     name->t.size += 2;
640     return name;
641 }

8.6.3.26 PublicMarshalAndComputeName()

This function computes the Name of an object from its public area.

642  TPM2B_NAME *
643  PublicMarshalAndComputeName(  
644     TPMT_PUBLIC *publicArea, // IN: public area of an object
645     TPM2B_NAME *name        // OUT: name of the object
646 )
647 {
648     // Will marshal a public area into a template. This is because the internal
649     // format for a TPM2B_PUBLIC is a structure and not a simple BYTE buffer.
650     TPM2B_TEMPLATE marshaled; // this is big enough to hold a
651     // marshaled TPMT_PUBLIC
652     BYTE *buffer = (BYTE *)&marshaled.t.buffer;
653     // if the nameAlg is NULL then there is no name.
654     if (publicArea->nameAlg == TPM_ALG_NULL)
655         name->t.size = 0;
656     else
657         { // Marshal the public area into its canonical form
marshaled.t.size = TPMT_PUBLIC_Marshal(publicArea, &buffer, NULL);

// and compute the name
ObjectComputeName(marshaled.t.size, marshaled.t.buffer,
    publicArea->nameAlg, name);

return name;

8.6.3.27 ComputeQualifiedName()

This function computes the qualified name of an object.

```c
void ComputeQualifiedName(
    TPM_HANDLE       parentHandle, // IN: parent's handle
    TPM_ALG_ID       nameAlg,     // IN: name hash
    TPM2B_NAME      *name,        // IN: name of the object
    TPM2B_NAME      *qualifiedName // OUT: qualified name of the object
)
{
    HASH_STATE hashState;    // hash state
    TPM2B_NAME parentName;

    // if(parentHandle == TPM_RH_UNASSIGNED)
    {
        MemoryCopy2B(&qualifiedName->b, &name->b, sizeof(qualifiedName->t.name));
        *qualifiedName = *name;
    }
    else
    {
        GetQualifiedName(parentHandle, &parentName);

        // QN_A = hash_A (QN of parent || NAME_A)
        // Start hash
        qualifiedName->t.size = CryptHashStart(&hashState, nameAlg);

        // Add parent's qualified name
        CryptDigestUpdate2B(&hashState, &parentName.b);

        // Add self name
        CryptDigestUpdate2B(&hashState, &name->b);

        // Complete hash leaving room for the name algorithm
        CryptHashEnd(&hashState, qualifiedName->t.size,
            &qualifiedName->t.name[2]);
        UINT16_TO_BYTE_ARRAY(nameAlg, qualifiedName->t.name);
        qualifiedName->t.size += 2;

        return;
    }
}
```

8.6.3.28 ObjectIsStorage()

This function determines if an object has the attributes associated with a parent. A parent is an asymmetric or symmetric block cipher key that has its restricted and decrypt attributes SET, and sign CLEAR.
### 8.6.3.29 ObjectCapGetLoaded()

This function returns a list of handles of loaded object, starting from `handle`. `Handle` must be in the range of valid transient object handles, but does not have to be the handle of a loaded transient object.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more handles available</td>
</tr>
<tr>
<td>NO</td>
<td>all the available handles has been returned</td>
</tr>
</tbody>
</table>

```c
TPMI_YES_NO ObjectCapGetLoaded(
    TPMI_DH_OBJECT handle, // IN: start handle
    UINT32 count, // IN: count of returned handles
    TPML_HANDLE *handleList // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    UINT32 i;
    // pAssert(HandleGetType(handle) == TPM_HT_TRANSIENT);
    // Initialize output handle list
    handleList->count = 0;
    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES) count = MAX_CAP_HANDLES;
    // Iterate object slots to get loaded object handles
    for(i = handle - TRANSIENT_FIRST; i < MAX_LOADED_OBJECTS; i++)
    {
        if(s_objects[i].attributes.occupied == TRUE)
        {
            // A valid transient object can not be the copy of a persistent object
            pAssert(s_objects[i].attributes.evict == CLEAR);
            if(handleList->count < count)
            {
                // If we have not filled up the return list, add this object
                // handle to it
                handleList->handle[handleList->count] = i + TRANSIENT_FIRST;
            }
        }
    }
    return more;
}
```
8.6.3.30 ObjectCapGetTransientAvail()

This function returns an estimate of the number of additional transient objects that could be loaded into the TPM.

```c
UINT32 ObjectCapGetTransientAvail()
{
    void
    {
        UINT32 i;
        UINT32 num = 0;
        // // Iterate object slot to get the number of unoccupied slots
        for(i = 0; i < MAX_LOADED_OBJECTS; i++)
            if(s_objects[i].attributes.occupied == FALSE) num++;
        return num;
    }
}
```

8.6.3.31 ObjectGetPublicAttributes()

Returns the attributes associated with an object handles.

```c
TPMA_OBJECT ObjectGetPublicAttributes(
    TPM_HANDLE handle
)
{
    return HandleToObject(handle)->publicArea.objectAttributes;
}
```

```c
OBJECT_ATTRIBUTES ObjectGetProperties(
    TPM_HANDLE handle
)
{
    return HandleToObject(handle)->attributes;
}
```
8.7 PCR.c

8.7.1 Introduction

This function contains the functions needed for PCR access and manipulation.

This implementation uses a static allocation for the PCR. The amount of memory is allocated based on the number of PCR in the implementation and the number of implemented hash algorithms. This is not the expected implementation. PCR SPACE DEFINITIONS.

In the definitions below, the _g_hashPcrMap_ is a bit array that indicates which of the PCR are implemented. The _g_hashPcr_ array is an array of digests. In this implementation, the space is allocated whether the PCR is implemented or not.

8.7.2 Includes, Defines, and Data Definitions

```c
#define PCR_C
#include "Tpm.h"

The initial value of PCR attributes. The value of these fields should be consistent with PC Client specification. In this implementation, we assume the total number of implemented PCR is 24.

static const PCR_Attributes s_initAttributes[] = {
    // PCR 0 - 15, static RTM
    {1, 0, 0x1F}, {1, 0, 0x1F}, {1, 0, 0x1F}, {1, 0, 0x1F},
    {1, 0, 0x1F}, {1, 0, 0x1F}, {1, 0, 0x1F}, {1, 0, 0x1F},
    {1, 0, 0x1F}, {1, 0, 0x1F}, {1, 0, 0x1F}, {1, 0, 0x1F},
    {1, 0, 0x1F}, {1, 0, 0x1F}, {1, 0, 0x1F}, {1, 0, 0x1F},
    {0, 0x0F, 0x1F}, // PCR 16, Debug
    {0, 0x10, 0x1C}, // PCR 17, Locality 4
    {0, 0x10, 0x1C}, // PCR 18,Locality 3
    {0, 0x10, 0x0C}, // PCR 19, Locality 2
    {0, 0x14, 0x0E}, // PCR 20, Locality 1
    {0, 0x14, 0x04}, // PCR 21, Dynamic OS
    {0, 0x14, 0x04}, // PCR 22, Dynamic OS
    {0, 0x0F, 0x1F}, // PCR 23, Application specific
    {0, 0x0F, 0x1F} // PCR 24, testing policy
};
```

8.7.3 Functions

8.7.3.1 PCRBelongsAuthGroup()

This function indicates if a PCR belongs to a group that requires an _authValue_ in order to modify the PCR. If it does, _groupId_ is set to value of the group index. This feature of PCR is decided by the platform specification.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>PCR belongs an authorization group</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>PCR does not belong an authorization group</td>
</tr>
</tbody>
</table>

```c
BOOL PCRBelongsAuthGroup(
    TPMI_DH_PCR handle, // IN: handle of PCR
    UINT32 *groupId    // OUT: group index if PCR belongs a
);```

8.7.3.2 PCRBelongsPolicyGroup()

This function indicates if a PCR belongs to a group that requires a policy authorization in order to modify the PCR. If it does, *groupIndex is set to value of the group index. This feature of PCR is decided by the platform specification.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>PCR belongs a policy group</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>PCR does not belong a policy group</td>
</tr>
</tbody>
</table>

```c
BOOL PCRBelongsPolicyGroup(
    TPMIDH_PCR handle,  // IN: handle of PCR
    UINT32 *groupIndex   // OUT: group index if PCR belongs a group that allows policy. If PCR does not belong to a policy group, the value in this parameter is invalid
){
    #if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
        // Platform specification decides if a PCR belongs to a policy group and belongs to which group. In this implementation, we assume there is only one policy group which contains PCR20-22. If the platform specification requires differently, the implementation should be changed accordingly
        if(handle >= 20 && handle <= 22)
        {
            *groupIndex = 0;
            return TRUE;
        }
    #endif
    return FALSE;
}
```

8.7.3.3 PCRBelongsTCBGroup()

This function indicates if a PCR belongs to the TCB group.

```c
{  
    #if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
        // Platform specification decides if a PCR belongs to a policy group and belongs to which group. In this implementation, we assume there is only one policy group which contains PCR20-22. If the platform specification requires differently, the implementation should be changed accordingly
        if(handle >= 20 && handle <= 22)
        {  
            *groupIndex = 0;
            return TRUE;
        }
    #endif
    return FALSE;
}
```
### 8.7.3.4 PCRPolicyIsAvailable()

This function indicates if a policy is available for a PCR.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>the PCR should be authorized by policy</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>the PCR does not allow policy</td>
</tr>
</tbody>
</table>

```c
SOOL
PCRPolicyIsAvailable TMD_DH_PCR handle  // IN: PCR handle
{
  UINT32 groupIndex;
  return PCRBelongsPolicyGroup(handle, &groupIndex);
}
```

### 8.7.3.5 PCRGetAuthValue()

This function is used to access the `authValue` of a PCR. If PCR does not belong to an `authValue` group, an `EmptyAuth()` will be returned.

```c
TPM2B_AUTH *
PCRGetAuthValue TMD_DH_PCR handle  // IN: PCR handle
{
  UINT32 groupIndex;
  if(PCRBelongsAuthGroup(handle, &groupIndex))
  {
    return &gc.pcrAuthValues.auth[groupIndex];
  }
  else
  {
    return NULL;
  }
```
8.7.3.6 PCRGetAuthPolicy()

This function is used to access the authorization policy of a PCR. It sets policy to the authorization policy and returns the hash algorithm for policy. If the PCR does not allow a policy, TPM_ALG_NULL is returned.

```c
TPMI_ALG_HASH
PCRGetAuthPolicy(
    TPMI_DH_PCR  handle,  // IN: PCR handle
    TPM2B_DIGEST *policy  // OUT: policy of PCR
)
{
    UINT32          groupIndex;
    if (PCRBelongsPolicyGroup(handle, &groupIndex))
    {
        *policy = gp.pcrPolicies.policy[groupIndex];
        return gp.pcrPolicies.hashAlg[groupIndex];
    }
    else
    {
        policy->t.size = 0;
        return TPM_ALG_NULL;
    }
}
```

8.7.3.7 PCRSimStart()

This function is used to initialize the policies when a TPM is manufactured. This function would only be called in a manufacturing environment or in a TPM simulator.

```c
void
PCRSimStart(
    void
){
    UINT32  i;
    #if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
    for(i = 0; i < NUM_POLICY_PCR_GROUP; i++)
    {  
        gp.pcrPolicies.hashAlg[i] = TPM_ALG_NULL;
        gp.pcrPolicies.policy[i].t.size = 0;
    }
    #endif
    #if defined NUM_AUTHVALUE_PCR_GROUP && NUM_AUTHVALUE_PCR_GROUP > 0
    for(i = 0; i < NUM_AUTHVALUE_PCR_GROUP; i++)
    {  
        gc.pcrAuthValues.auth[i].t.size = 0;
    }
    #endif
    // We need to give an initial configuration on allocated PCR before
    // receiving any TPM2_PCR_Allocate command to change this configuration
    // When the simulation environment starts, we allocate all the PCRs
    for(gp.pcrAllocated.count = 0; gp.pcrAllocated.count < HASH_COUNT;
        gp.pcrAllocated.count++)
    {  
        gp.pcrAllocated.pcrSelections[gp.pcrAllocated.count].hash
            = CryptHashGetAlgByIndex(gp.pcrAllocated.count);
        gp.pcrAllocated.pcrSelections[gp.pcrAllocated.count].sizeofSelect
```
8.7.3.8 GetSavedPcrPointer()

This function returns the address of an array of state saved PCR based on the hash algorithm.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>no such algorithm</td>
</tr>
<tr>
<td>!= NULL</td>
<td>pointer to the 0th byte of the 0th PCR</td>
</tr>
</tbody>
</table>

```c
static BYTE *
GetSavedPcrPointer(
    TPM_ALG_ID       alg,       // IN: algorithm for bank
    UINT32           pcrIndex    // IN: PCR index in PCR_SAVE
)
{
    switch(alg)
    {
        #if ALG_SHA1
            case ALG_SHA1_VALUE:
                return gc.pcrSave.sha1[pcrIndex];
                break;
        #endif
        #if ALG_SHA256
            case ALG_SHA256_VALUE:
                return gc.pcrSave.sha256[pcrIndex];
                break;
        #endif
        #if ALG_SHA384
            case ALG_SHA384_VALUE:
                return gc.pcrSave.sha384[pcrIndex];
                break;
        #endif
        #if ALG_SHA512
            case ALG_SHA512_VALUE:
                return gc.pcrSave.sha512[pcrIndex];
                break;
        #endif
        #if ALG_SM3_256
            case ALG_SM3_256_VALUE:
                return gc.pcrSave.sm3_256[pcrIndex];
                break;
        #endif
        default:
            break;
    }
    FAIL(FATAL_ERROR_INTERNAL);
}
```
### 8.7.3.9 PcrIsAllocated()

This function indicates if a PCR number for the particular hash algorithm is allocated.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>PCR is allocated</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>PCR is not allocated</td>
</tr>
</tbody>
</table>

```cpp
BOOL PcrIsAllocated(
    UINT32 pcr,       // IN: The number of the PCR
    TPMI_ALG_HASH hashAlg // IN: The PCR algorithm
)
{
    UINT32 i;
    BOOL allocated = FALSE;

    if (pcr < IMPLEMENTATION_PCR)
        for (i = 0; i < gp.pcrAllocated.count; i++)
            if (gp.pcrAllocated.pcrSelections[i].hash == hashAlg)
                if (((gp.pcrAllocated.pcrSelections[i].pcrSelect[pcr / 8]) & (1 << (pcr % 8))) != 0)
                    allocated = TRUE;
                else
                    allocated = FALSE;
                break;
            return allocated;
```

### 8.7.3.10 GetPcrPointer()

This function returns the address of an array of PCR based on the hash algorithm.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>no such algorithm</td>
</tr>
<tr>
<td>!NULL</td>
<td>pointer to the 0th byte of the 0th PCR</td>
</tr>
</tbody>
</table>

```cpp
static BYTE * GetPcrPointer(
    TPM_ALG_ID alg,       // IN: algorithm for bank
    UINT32 pcrNumber      // IN: PCR number
)
{
    static BYTE *pcr = NULL;

    if (!PcrIsAllocated(pcrNumber, alg))
        return NULL;

    switch(alg)
    {
    #if ALG_SHA1
        case ALG_SHA1_VALUE:
            pcr = s_pcrs[pcrNumber].sha1Pcr;
            break;
    ```
8.7.3.11 IsPcrSelected()

This function indicates if an indicated PCR number is selected by the bit map in selection.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>PCR is selected</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>PCR is not selected</td>
</tr>
</tbody>
</table>

static BOOL IsPcrSelected(
    UINT32 pcr,       // IN: The number of the PCR
    TPMS_PCR_SELECTION *selection  // IN: The selection structure
)
{}
    BOOL selected;
    selected = (pcr < IMPLEMENTATION_PCR
        &
        ((selection->pcrSelect[pcr / 8]) & (1 << (pcr % 8))) != 0);
    return selected;

8.7.3.12 FilterPcr()

This function modifies a PCR selection array based on the implemented PCR.

static void FilterPcr(
    TPMS_PCR_SELECTION *selection  // IN: input PCR selection
)
{}
    UINT32 i;
    TPMS_PCR_SELECTION *allocated = NULL;
// If size of select is less than PCR_SELECT_MAX, zero the unspecified PCR
for(i = selection->sizeofSelect; i < PCR_SELECT_MAX; i++)
    selection->pcrSelect[i] = 0;

// Find the internal configuration for the bank
for(i = 0; i < gp.pcrAllocated.count; i++)
    if(gp.pcrAllocated.pcrSelections[i].hash == selection->hash)
        allocated = &gp.pcrAllocated.pcrSelections[i];
    break;

for(i = 0; i < selection->sizeofSelect; i++)
    if(allocated == NULL)
        // If the required bank does not exist, clear input selection
        selection->pcrSelect[i] = 0;
    else
        selection->pcrSelect[i] &= allocated->pcrSelect[i];

return;

8.7.3.13 PcrDrtm()

This function does the DRTM and H-CRTM processing it is called from _TPM_Hash_End().

void PcrDrtm(
    const TPMI_DH_PCR pcrHandle,  // IN: the index of the PCR to be
    const TPMI_ALG_HASH hash,     // IN: the bank identifier
    const TPM2B_DIGEST *digest    // IN: the digest to modify the PCR
)
{
    BYTE       *pcrData = GetPcrPointer(hash, pcrHandle);

    if(pcrData != NULL)
    {
        // Rest the PCR to zeros
        MemorySet(pcrData, 0, digest->t.size);

        // if the TPM has not started, then set the PCR to 0...04 and then extend
        if(!TPMIsStarted())
            pcrData[digest->t.size - 1] = 4;

        // Now, extend the value
        PCRExtend(pcrHandle, hash, digest->t.size, (BYTE *)digest->t.buffer);
    }

8.7.3.14 PCR_ClearAuth()

This function is used to reset the PCR authorization values. It is called on TPM2_Startup(CLEAR) and
TPM2_Clear().

void
Part 4: Supporting Routines

Trusted Platform Module Library

8.7.3.15 PCRStartup()

This function initializes the PCR subsystem at TPM2_Startup().

```c
void

PCRStartup(
    STARTUP_TYPE     type,       // IN: startup type
    BYTE             locality,   // IN: startup locality
)
{
    UINT32              pcr, j;
    UINT32              saveIndex = 0;

    g_pcrReConfig = FALSE;

    // Don’t test for SU_RESET because that should be the default when nothing
    // else is selected
    if(type != SU_RESUME && type != SU_RESTART)
    {
        // PCR generation counter is cleared at TPM_RESET
        gr.pcrCounter = 0;
    }

    // Initialize/Restore PCR values
    for(pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
    {
        // On resume, need to know if this PCR had its state saved or not
        UINT32 stateSaved;

        if(type == SU_RESUME
            && s_initAttributes[pcr].stateSave == SET)
        {
            stateSaved = 1;
        }
        else
        {
            stateSaved = 0;
            PCRChanged(pcr);
        }

        // If this is the H-CRTM PCR and we are not doing a resume and we
        // had an H-CRTM event, then we don’t change this PCR
        if(pcr == HCRTM_PCR && type != SU_RESUME && g_DrtmPreStartup == TRUE)
            continue;

        // Iterate each hash algorithm bank
        for(j = 0; j < gp.pcrAllocated.count; j++)
        {
            TPMI_ALG_HASH    hash = gp.pcrAllocated.pcrSelections[j].hash;
            BYTE             *pcrData = GetPcrPointer(hash, pcr);
            UINT16           pcrSize = CryptHashGetDigestSize(hash);
        }
    }
}
if (pcrData != NULL) {
    if (stateSaved == 1) {
        // Restore saved PCR value
        BYTE *pcrSavedData = GetSavedPcrPointer(
            gp.pcrAllocated.pcrSelections[j].hash,
            saveIndex);
        MemoryCopy(pcrData, pcrSavedData, pcrSize);
    } else {
        // PCR was not restored by state save
        if ((s_initAttributes[pcr].resetLocality & 0x10) != 0)
            MemorySet(pcrData, 0xFF, pcrSize);
        else {
            MemorySet(pcrData, 0, pcrSize);
            if (pcr == HCRTM_PCR)
                pcrData[pcrSize - 1] = locality;
        }
    }
}

8.7.3.16 PCRStateSave()

This function is used to save the PCR values that will be restored on TPM Resume.

```c
void PCRStateSave(TPM_SU type) {
    UINT32 pcr, j;
    UINT32 saveIndex = 0;

    // if state save CLEAR, nothing to be done. Return here
    if (type == TPM_SU_CLEAR)
        return;

    // Copy PCR values to the structure that should be saved to NV
    for (pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++) {
        UINT32 stateSaved = (s_initAttributes[pcr].stateSave == SET) ? 1 : 0;

        // Iterate each hash algorithm bank
        for (j = 0; j < gp.pcrAllocated.count; j++) {
            BYTE *pcrData;
            UINT32 pcrSize;
            ...
pcrData = GetPcrPointer(gp.pcrAllocated.pcrSelections[j].hash, pcr);

if (pcrData != NULL) {
    pcrSize = CryptHashGetDigestSize(gp.pcrAllocated.pcrSelections[j].hash);
    if (stateSaved == 1) {
        // Restore saved PCR value
        BYTE *pcrSavedData = GetSavedPcrPointer(gp.pcrAllocated.pcrSelections[j].hash, saveIndex);
        MemoryCopy(pcrSavedData, pcrData, pcrSize);
    }
    saveIndex += stateSaved;
}
return;

8.7.3.17 PCRIsStateSaved()

This function indicates if the selected PCR is a PCR that is state saved on TPM2_Shutdown(STATE). The return value is based on PCR attributes.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>PCR is state saved</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>PCR is not state saved</td>
</tr>
</tbody>
</table>

BOOL PCRIsStateSaved(
    TPMI_DH_PCR handle       // IN: PCR handle to be extended
) {
    UINT32 pcr = handle - PCR_FIRST;
    if (s_initAttributes[pcr].stateSave == SET)
        return TRUE;
    else
        return FALSE;
}

8.7.3.18 PCRIsResetAllowed()

This function indicates if a PCR may be reset by the current command locality. The return value is based on PCR attributes, and not the PCR allocation.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>TPM2_PCR_Reset() is allowed</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>TPM2_PCR_Reset() is not allowed</td>
</tr>
</tbody>
</table>

BOOL PCRIsResetAllowed(
    TPMI_DH_PCR handle       // IN: PCR handle to be extended
)
8.7.3.19 PCRChanged()

This function checks a PCR handle to see if the attributes for the PCR are set so that any change to the PCR causes an increment of the pcrCounter. If it does, then the function increments the counter. Will also bump the counter if the handle is zero which means that PCR 0 cannot be in the TCB group. Bump on zero is used by TPM2_Clear().

```c
void
PCRChanged(
    TPM_HANDLE       pcrHandle   // IN: the handle of the PCR that changed.
)
{
    // For the reference implementation, the only change that does not cause
    // increment is a change to a PCR in the TCB group.
    if((pcrHandle == 0) || !PCRBelongsTCBGroup(pcrHandle))
    {
        gr.pcrCounter++;
        if(gr.pcrCounter == 0)
            FAIL(FATAL_ERROR_COUNTER_OVERFLOW);
    }
}
```

8.7.3.20 PCRIsExtendAllowed()

This function indicates a PCR may be extended at the current command locality. The return value is based on PCR attributes, and not the PCR allocation.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>extend is allowed</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>extend is not allowed</td>
</tr>
</tbody>
</table>

```c
BOOL
PCRIsExtendAllowed(
    TPMI_DH_PCR      handle   // IN: PCR handle to be extended
)
{
    UINT8               commandLocality;
    UINT8               localityBits = 1;
    UINT32              pcr = handle - PCR_FIRST;
```

{ }
8.7.3.21 PCRExtend()

This function is used to extend a PCR in a specific bank.

```c
void PCRExtend(
    TPMI_DH_PCR handle, // IN: PCR handle to be extended
    TPMI_ALG_HASH hash, // IN: hash algorithm of PCR
    UINT32 size, // IN: size of data to be extended
    BYTE *data // IN: data to be extended
)
{
    BYTE *pcrData;
    HASH_STATE hashState;
    UINT16 pcrSize;
    pcrData = GetPcrPointer(hash, handle - PCR_FIRST);
    if(pcrData != NULL)
    {
        pcrSize = CryptHashGetDigestSize(hash);
        CryptHashStart(&hashState, hash);
        CryptDigestUpdate(&hashState, pcrSize, pcrData);
        CryptDigestUpdate(&hashState, size, data);
        CryptHashEnd(&hashState, pcrSize, pcrData);
        // PCR has changed so update the pcrCounter if necessary
        PCRChanged(handle);
    }
    return;
}
```

8.7.3.22 PCRComputeCurrentDigest()

This function computes the digest of the selected PCR.

As a side-effect, `selection` is modified so that only the implemented PCR will have their bits still set.

```c
void PCRComputeCurrentDigest(
    TPMI_ALG_HASH hashAlg, // IN: hash algorithm to compute digest
    TPML_PCR_SELECTION *selection, // IN/OUT: PCR selection (filtered on output)
    TPM2B_DIGEST *digest // OUT: digest
)
{
    HASH_STATE hashState;
    TPMs_PCR_SELECTION *select;
    BYTE *pcrData; // will point to a digest
    UINT32 pcrSize;
    UINT32 pcr;
```
UINT32 i;

// Initialize the hash
digest->t.size = CryptHashStart(&hashState, hashAlg);
pAssert(digest->t.size > 0 && digest->t.size < UINT16_MAX);

// Iterate through the list of PCR selection structures
for (i = 0; i < selection->count; i++)
{
    // Point to the current selection
    select = &selection->pcrSelections[i]; // Point to the current selection
    FilterPcr(select); // Clear out the bits for unimplemented PCR

    // Need the size of each digest
    pcrSize = CryptHashGetDigestSize(selection->pcrSelections[i].hash);

    // Iterate through the selection
    for (pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
    {
        if (IsPcrSelected(pcr, select)) // Is this PCR selected
        {
            // Get pointer to the digest data for the bank
            pcrData = GetPcrPointer(selection->pcrSelections[i].hash, pcr);
            pAssert(pcrData != NULL);
            CryptDigestUpdate(&hashState, pcrSize, pcrData); // add to digest
        }  
    }
}

// Complete hash stack
CryptHashEnd2B(&hashState, &digest->b);
return;

8.7.3.23 PCRRead()

This function is used to read a list of selected PCR. If the requested PCR number exceeds the maximum number that can be output, the selection is adjusted to reflect the actual output PCR.

void PCRRead(
    TPML_PCR_SELECTION *selection, // IN/OUT: PCR selection (filtered on
    TPML_DIGEST         *digest,  // OUT: digest
    UINT32              *pcrCounter // OUT: the current value of PCR generation number
)
{
    TPMS_PCR_SELECTION *select;
    BYTE               *pcrData;  // will point to a digest
    UINT32             pcr;
    UINT32             i;

    digest->count = 0;

    // Iterate through the list of PCR selection structures
    for (i = 0; i < selection->count; i++)
    {
        // Point to the current selection
        select = &selection->pcrSelections[i]; // Point to the current selection
        FilterPcr(select); // Clear out the bits for unimplemented PCR

        // Iterate through the selection
        for (pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
```c
{ 
  if(IsPcrSelected(pcr, select)) // Is this PCR selected
  {
    // Check if number of digest exceed upper bound
    if(digest->count > 7)
    {
      // Clear rest of the current select bitmap
      while(pcr < IMPLEMENTATION_PCR
        // do not round up!
        && (pcr / 8) < select->sizeofSelect)
        {
          // do not round up!
          select->pcrSelect[pcr / 8] &= (BYTE)~(1 << (pcr % 8));
          pcr++;
        }
      // Exit inner loop
      break;
    }
  // Need the size of each digest
  digest->digests[digest->count].t.size = 
    CryptHashGetDigestSize(selection->pcrSelections[i].hash);
  // Get pointer to the digest data for the bank
  pcrData = GetPcrPointer(selection->pcrSelections[i].hash, pcr);
  pAssert(pcrData != NULL);
  // Add to the data to digest
  MemoryCopy(digest->digests[digest->count].t.buffer, 
            pcrData, 
            digest->digests[digest->count].t.size);
  digest->count++;
}
// If we exit inner loop because we have exceed the output upper bound
if(digest->count > 7 && pcr < IMPLEMENTATION_PCR)
{
  // Clear rest of the selection
  while(i < selection->count)
  {
    MemorySet(selection->pcrSelections[i].pcrSelect, 0, 
              selection->pcrSelections[i].sizeofSelect);
    i++;
  }
  // exit outer loop
  break;
}
}

8.7.3.24  PcrWrite()

This function is used by _TPM_Hash_End() to set a PCR to the computed hash of the H-CRTM event.

```
BYTE *pcrData;

// Copy value to the PCR if it is allocated
pcrData = GetPcrPointer(hash, pcr);
if (pcrData != NULL)
{
    MemoryCopy(pcrData, digest->t.buffer, digest->t.size);
}
return;

8.7.3.25 PCRAssign()

This function is used to change the PCR allocation.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>allocate failed</td>
</tr>
<tr>
<td>TPM_RC_PCR</td>
<td>improper allocation</td>
</tr>
</tbody>
</table>

TPM_RC PCRAssign(
    TPML_PCR_SELECTION *allocate, // IN: required allocation
    UINT32 *maxPCR, // OUT: Maximum number of PCR
    UINT32 *sizeNeeded, // OUT: required space
    UINT32 *sizeAvailable // OUT: available space
)
{
    UINT32 i, j, k;
    TPML_PCR_SELECTION newAllocate;
    // Initialize the flags to indicate if HCRTM PCR and DRTM PCR are allocated.
    BOOL pcrHcrtc = FALSE;
    BOOL pcrDrtm = FALSE;
    // Create the expected new PCR allocation based on the existing allocation
    // and the new input:
    // 1. if a PCR bank does not appear in the new allocation, the existing
    // allocation of this PCR bank will be preserved.
    // 2. if a PCR bank appears multiple times in the new allocation, only the
    // last one will be in effect.
    newAllocate = gp.pcrAllocated;
    for (i = 0; i < allocate->count; i++)
    {
        for (j = 0; j < newAllocate.count; j++)
        {
            // If hash matches, the new allocation covers the old allocation
            // for this particular bank.
            // The assumption is the initial PCR allocation (from manufacture)
            // has all the supported hash algorithms with an assigned bank
            // (possibly empty). So there must be a match for any new bank
            // allocation from the input.
            if (newAllocate.pcrSelections[j].hash ==
                allocate->pcrSelections[i].hash)
            {
                newAllocate.pcrSelections[j] = allocate->pcrSelections[i];
                break;
            }
        }
    }
    // The j loop must exit with a match.
    pAssert(j < newAllocate.count);
// Max PCR in a bank is MIN(implemented PCR, PCR with attributes defined)
*maxPCR = sizeof(s_initAttributes) / sizeof(PCR_Attributes);
if(*maxPCR > IMPLEMENTATION_PCR)
*maxPCR = IMPLEMENTATION_PCR;

// Compute required size for allocation
*sizeNeeded = 0;
for(i = 0; i < newAllocate.count; i++)
{
UINT32 digestSize = CryptHashGetDigestSize(newAllocate.pcrSelections[i].hash);
#if defined(DRTM_PCR)
// Make sure that we end up with at least one DRTM PCR
pcrDrtm = pcrDrtm || TestBit(DRTM_PCR,
   newAllocate.pcrSelections[i].pcrSelect,
   newAllocate.pcrSelections[i].sizeofSelect);
#else
// if DRTM PCR is not required, indicate that the allocation is OK
pcrDrtm = TRUE;
#endif
#if defined(HCRTM_PCR)
// and one HCRTM PCR (since this is usually PCR 0...)
pcrHcrtm = pcrHcrtm || TestBit(HCRTM_PCR,
   newAllocate.pcrSelections[i].pcrSelect,
   newAllocate.pcrSelections[i].sizeofSelect);
#else
pcrHcrtm = TRUE;
#endif
for(j = 0; j < newAllocate.pcrSelections[i].sizeofSelect; j++)
{
BYTE mask = 1;
for(k = 0; k < 8; k++)
{
   if((newAllocate.pcrSelections[i].pcrSelect[j] & mask) != 0)
      *sizeNeeded += digestSize;
   mask = mask << 1;
}
}
if(!pcrDrtm || !pcrHcrtm)
   return TPM_RC_PCR;

// In this particular implementation, we always have enough space to
// allocate PCR. Different implementation may return a sizeAvailable less
// than the sizeNeed.
*sizeAvailable = sizeof(s_pcrs);

// Save the required allocation to NV. Note that after NV is written, the
// PCR allocation in NV is no longer consistent with the RAM data
// gp.pcrAllocated. The NV version reflect the allocate after next
// TPM_RESET, while the RAM version reflects the current allocation
NV_WRITE_PERSISTENT(pcrAllocated, newAllocate);
return TPM_RC_SUCCESS;
}

8.7.3.26  PCRSetSetValue()

This function is used to set the designated PCR in all banks to an initial value. The initial value is signed
and will be sign extended into the entire PCR.

void
```c
PCRSetValue(
    TPM_HANDLE handle, // IN: the handle of the PCR to set
    INT8 initialValue // IN: the value to set
)
{
    int i;
    UINT32 pcr = handle - PCR_FIRST;
    TPMI_ALG_HASH hash;
    UINT16 digestSize;
    BYTE *pcrData;

    // Iterate supported PCR bank algorithms to reset
    for(i = 0; i < HASH_COUNT; i++)
    {
        hash = CryptHashGetAlgByIndex(i);
        // Prevent runaway
        if(hash == TPM_ALG_NULL)
            break;

        // Get a pointer to the data
        pcrData = GetPcrPointer(gp.pcrAllocated.pcrSelections[i].hash, pcr);

        // If the PCR is allocated
        if(pcrData != NULL)
        {
            // And the size of the digest
            digestSize = CryptHashGetDigestSize(hash);

            // Set the LSO to the input value
            pcrData[digestSize - 1] = initialValue;

            // Sign extend
            if(initialValue >= 0)
                MemorySet(pcrData, 0, digestSize - 1);
            else
                MemorySet(pcrData, -1, digestSize - 1);
        }
    }
}

8.7.3.27 PCRResetDynamics

This function is used to reset a dynamic PCR to 0. This function is used in DRTM sequence.

void
PCRResetDynamics(
    void
)
{
    UINT32 pcr, i;
    // Initialize PCR values
    for(pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
    {
        // Iterate each hash algorithm bank
        for(i = 0; i < gp.pcrAllocated.count; i++)
        {
            BYTE *pcrData;
            UINT32 pcrSize;

            pcrData = GetPcrPointer(gp.pcrAllocated.pcrSelections[i].hash, pcr);

            if(pcrData != NULL)
                {
```
8.7.3.28 PCRCapGetAllocation()

This function is used to get the current allocation of PCR banks.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if the return count is 0</td>
</tr>
<tr>
<td>NO</td>
<td>if the return count is not 0</td>
</tr>
</tbody>
</table>

```c
TPMI_YES_NO
PCRCapGetAllocation(
    UINT32               count, // IN: count of return
    TPML_PCR_SELECTION  *pcrSelection // OUT: PCR allocation list
)
{
    if(count == 0)
    {
        pcrSelection->count = 0;
        return YES;
    }
    else
    {
        *pcrSelection = gp.pcrAllocated;
        return NO;
    }
}
```

8.7.3.29 PCRSetSelectBit()

This function sets a bit in a bitmap array.

```c
static void
PCRSetSelectBit(
    UINT32           pcr, // IN: PCR number
    BYTE            *bitmap // OUT: bit map to be set
)
{
    bitmap[pcr / 8] |= (1 << (pcr % 8));
    return;
}
```

8.7.3.30 PCRGetProperty()

This function returns the selected PCR property.
Return Value | Meaning
-------------|-----------------------------------
TRUE(1)      | the property type is implemented
FALSE(0)     | the property type is not implemented

```c
static BOOL PCRGetProperty(
    TPM_PT_PCR property,
    TPM_TAGGED_PCR_SELECT *select
)
{
    UINT32 pcr;
    UINT32 groupIndex;

    select->tag = property;
    // Always set the bitmap to be the size of all PCR
    select->sizeofSelect = (IMPLEMENTATION_PCR + 7) / 8;
    // Initialize bitmap
    MemorySet(select->pcrSelect, 0, select->sizeofSelect);
    // Collecting properties
    for (pcr = 0; pcr < IMPLEMENTATION_PCR; pcr++)
    {
        switch (property)
        {
            case TPM_PT_PCR_SAVE:
                if (s_initAttributes[pcr].stateSave == SET)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L0:
                if ((s_initAttributes[pcr].extendLocality & 0x01) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L0:
                if ((s_initAttributes[pcr].resetLocality & 0x01) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L1:
                if ((s_initAttributes[pcr].extendLocality & 0x02) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L1:
                if ((s_initAttributes[pcr].resetLocality & 0x02) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L2:
                if ((s_initAttributes[pcr].extendLocality & 0x04) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L2:
                if ((s_initAttributes[pcr].resetLocality & 0x04) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L3:
                if ((s_initAttributes[pcr].extendLocality & 0x08) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_RESET_L3:
                if ((s_initAttributes[pcr].resetLocality & 0x08) != 0)
                    PCRSetSelectBit(pcr, select->pcrSelect);
                break;
            case TPM_PT_PCR_EXTEND_L4:
                if ((s_initAttributes[pcr].extendLocality & 0x10) != 0)
```
984    PCRSetSelectBit(pcr, select->pcrSelect);
985    break;
986    case TPM_PT_PCR_RESET_L4:
987        if ((s_initAttributes[pcr].resetLocality & 0x10) != 0)
988            PCRSetSelectBit(pcr, select->pcrSelect);
989            break;
990    case TPM_PT_PCR_DRTM_RESET:
991        // DRTM reset PCRs are the PCR reset by locality 4
992        if ((s_initAttributes[pcr].resetLocality & 0x10) != 0)
993            PCRSetSelectBit(pcr, select->pcrSelect);
994        break;
995    #if defined NUM_POLICY_PCR_GROUP && NUM_POLICY_PCR_GROUP > 0
996    case TPM_PT_PCR_POLICY:
997        if (PCRBelongsPolicyGroup(pcr + PCR_FIRST, &groupIndex))
998            PCRSetSelectBit(pcr, select->pcrSelect);
999        break;
1000   #endif
1001   #if defined NUM_AUTHVALUE_PCR_GROUP && NUM_AUTHVALUE_PCR_GROUP > 0
1002    case TPM_PT_PCR_AUTH:
1003        if (PCRBelongsAuthGroup(pcr + PCR_FIRST, &groupIndex))
1004            PCRSetSelectBit(pcr, select->pcrSelect);
1005        break;
1006   #endif
1007   #if ENABLE_PCR_NO_INCREMENT == YES
1008    case TPM_PT_PCR_NO_INCREMENT:
1009        if (PCRBelongsTCBGroup(pcr + PCR_FIRST))
1010            PCRSetSelectBit(pcr, select->pcrSelect);
1011        break;
1012   #endif
1013    default:
1014        // If property is not supported, stop scanning PCR attributes
1015        // and return.
1016        return FALSE;
1017        break;
1018    }
1019    return TRUE;
1020 }

8.7.3.31 PCRCapGetProperties()

This function returns a list of PCR properties starting at property.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if no more property is available</td>
</tr>
<tr>
<td>NO</td>
<td>if there are more properties not reported</td>
</tr>
</tbody>
</table>

1022      TPMI_YES_NO
1023      PCRCapGetProperties(
1024      TPM_PT_PCR property,      // IN: the starting PCR property
1025      UINT32 count,            // IN: count of returned properties
1026      TPMML_TAGGED_PCR_PROPERTY *select  // OUT: PCR select
1027      )
1028      {
1029          TPMI_YES_NO more = NO;
1030          UINT32 i;
1031      }
1032      // Initialize output property list
1033      select->count = 0;
1034      // The maximum count of properties we may return is MAX_PCR_PROPERTIES
1035      if (count > MAX_PCR_PROPERTIES) count = MAX_PCR_PROPERTIES;
// TPM_PT_PCR_FIRST is defined as 0 in spec. It ensures that property
// value would never be less than TPM_PT_PCR_FIRST
caAssert(TPM_PT_PCR_FIRST == 0);

// Iterate PCR properties. TPM_PT_PCR_LAST is the index of the last property
// implemented on the TPM.
for(i = property; i <= TPM_PT_PCR_LAST; i++)
{
    if(select->count < count)
    {
        // If we have not filled up the return list, add more properties to it
        if(PCRGetProperty(i, &select->pcrProperty[select->count]))
            select->count++;
    }
    else
    {
        // If the return list is full but we still have properties
        // available, report this and stop iterating.
        more = YES;
        break;
    }
}
return more;

8.7.3.32 PCRCapGetHandles()

This function is used to get a list of handles of PCR, started from handle. If handle exceeds the maximum PCR handle range, an empty list will be returned and the return value will be NO.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more handles available</td>
</tr>
<tr>
<td>NO</td>
<td>all the available handles has been returned</td>
</tr>
</tbody>
</table>

TPMI_YES_NO
PCRCapGetHandles(
    TPMI_DH_PCR handle, // IN: start handle
    UINT32 count, // IN: count of returned handles
    TPML_HANDLE *handleList // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    UINT32 i;
    pAssert(HandleGetType(handle) == TPM_HT_PCR);

    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES) count = MAX_CAP_HANDLES;

    // Iterate PCR handle range
    for(i = handle & HR_HANDLE_MASK; i <= PCR_LAST; i++)
    {
        if(handleList->count < count)
        {
            // If we have not filled up the return list, add this PCR
            handleList->handle[handleList->count] = i + PCR_FIRST;
handleList->count++;

else
{
  // If the return list is full but we still have PCR handle
  // available, report this and stop iterating
  more = YES;
  break;
}

return more;
8.8 PP.c

8.8.1 Introduction

This file contains the functions that support the physical presence operations of the TPM.

8.8.2 Includes

```c
#include "Tpm.h"
```

8.8.3 Functions

8.8.3.1 PhysicalPresencePreInstall_Init()

This function is used to initialize the array of commands that always require confirmation with physical presence. The array is an array of bits that has a correspondence with the command code.

This command should only ever be executable in a manufacturing setting or in a simulation. When set, these cannot be cleared.

```c
void PhysicalPresencePreInstall_Init()
{
    COMMAND_INDEX commandIndex;
    // Clear all the PP commands
    MemorySet(&gp.ppList, 0, sizeof(gp.ppList));

    // Any command that is PP_REQUIRED should be SET
    for(commandIndex = 0; commandIndex < COMMAND_COUNT; commandIndex++)
    {
        if(s_commandAttributes[commandIndex] & IS_IMPLEMENTED
            && s_commandAttributes[commandIndex] & PP_REQUIRED)
            SET_BIT(commandIndex, gp.ppList);
    }

    // Write PP list to NV
    NV_SYNC_PERSISTENT(ppList);
    return;
}
```

8.8.3.2 PhysicalPresenceCommandSet()

This function is used to set the indicator that a command requires PP confirmation.

```c
void PhysicalPresenceCommandSet(
    TPM_CC commandCode // IN: command code
)
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);
    // if the command isn't implemented, do nothing
    if(commandIndex == UNIMPLEMENTED_COMMAND_INDEX)
        return;

    // only set the bit if this is a command for which PP is allowed
    if(s_commandAttributes[commandIndex] & PP_COMMAND)
```
8.8.3.3 PhysicalPresenceCommandClear()

This function is used to clear the indicator that a command requires PP confirmation.

```c
void
PhysicalPresenceCommandClear(
    TPM_CC commandCode // IN: command code
)
{
    COMMAND_INDEX commandIndex = CommandCodeToCommandIndex(commandCode);

    // If the command isn't implemented, then don't do anything
    if(commandIndex == UNIMPLEMENTED_COMMAND_INDEX)
        return;

    // Only clear the bit if the command does not require PP
    if((s_commandAttributes[commandIndex] & PP_REQUIRED) == 0)
        CLEAR_BIT(commandIndex, gp.ppList);

    return;
}
```

8.8.3.4 PhysicalPresenceIsRequired()

This function indicates if PP confirmation is required for a command.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>physical presence is required</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>physical presence is not required</td>
</tr>
</tbody>
</table>

```c
BOOL
PhysicalPresenceIsRequired(
    COMMAND_INDEX commandIndex // IN: command index
)
{
    // Check the bit map. If the bit is SET, PP authorization is required
    return (TEST_BIT(commandIndex, gp.ppList));
}
```

8.8.3.5 PhysicalPresenceCapGetCCList()

This function returns a list of commands that require PP confirmation. The list starts from the first implemented command that has a command code that the same or greater than commandCode.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more command codes available</td>
</tr>
<tr>
<td>NO</td>
<td>all the available command codes have been returned</td>
</tr>
</tbody>
</table>

```c
TPMI_YES_NO
PhysicalPresenceCapGetCCList(
    TPM_CC commandCode, // IN: start command code
    UINT32 count,       // IN: count of returned TPM_CC
    TPML_CC *commandList // OUT: list of TPM_CC
)
```
(TPMI_YES_NO more = NO;
COMMAND_INDEX commandIndex;

// Initialize output handle list
commandList->count = 0;

// The maximum count of command we may return is MAX_CAP_CC
if(count > MAX_CAP_CC) count = MAX_CAP_CC;

// Collect PP commands
for(commandIndex = GetClosestCommandIndex(commandCode);
commandIndex != UNIMPLEMENTED_COMMAND_INDEX;
commandIndex = GetNextCommandIndex(commandIndex))
{
    if(PhysicalPresenceIsRequired(commandIndex))
    {
        if(commandList->count < count)
        {
            // If we have not filled up the return list, add this command
            // code to it
            commandList->commandCodes[commandList->count] = GetCommandCode(commandIndex);
            commandList->count++;
        }
        else
        {
            // If the return list is full but we still have PP command
            // available, report this and stop iterating
            more = YES;
            break;
        }
    }
}
return more;
}
8.9 Session.c

8.9.1 Introduction

The code in this file is used to manage the session context counter. The scheme implemented here is a "truncated counter". This scheme allows the TPM to not need TPM_SU_CLEAR for a very long period of time and still not have the context count for a session repeated.

The counter (contextCounter) in this implementation is a UINT64 but can be smaller. The "tracking array" (contextArray) only has 16-bits per context. The tracking array is the data that needs to be saved and restored across TPM_SU_STATE so that sessions are not lost when the system enters the sleep state. Also, when the TPM is active, the tracking array is kept in RAM making it important that the number of bytes for each entry be kept as small as possible.

The TPM prevents collisions of these truncated values by not allowing a contextID to be assigned if it would be the same as an existing value. Since the array holds 16 bits, after a context has been saved, an additional \(2^{16}-1\) contexts may be saved before the count would again match. The normal expectation is that the context will be flushed before its count value is needed again but it is always possible to have long-lived sessions.

The contextID is assigned when the context is saved (TPM2_ContextSave()). At that time, the TPM will compare the low-order 16 bits of contextCounter to the existing values in contextArray and if one matches, the TPM will return TPM_RC_CONTEXT_GAP (by construction, the entry that contains the matching value is the oldest context).

The expected remediation by the TRM is to load the oldest saved session context (the one found by the TPM), and save it. Since loading the oldest session also eliminates its contextID value from contextArray, there TPM will always be able to load and save the oldest existing context.

In the worst case, software may have to load and save several contexts in order to save an additional one. This should happen very infrequently.

When the TPM searches contextArray and finds that none of the contextIDs match the low-order 16-bits of contextCount, the TPM can copy the low bits to the contextArray associated with the session, and increment contextCount.

There is one entry in contextArray for each of the active sessions allowed by the TPM implementation. This array contains either a context count, an index, or a value indicating the slot is available (0).

The index into the contextArray is the handle for the session with the region selector byte of the session set to zero. If an entry in contextArray contains 0, the corresponding handle may be assigned to a session. If the entry contains a value that is less than or equal to the number of loaded sessions for the TPM, then the array entry is the slot in which the context is loaded.

**EXAMPLE:**

If the TPM allows 8 loaded sessions, then the slot numbers would be 1-8 and a contextArray value in that range would represent the loaded session.

**NOTE:**

When the TPM firmware determines that the array entry is for a loaded session, it will subtract 1 to create the zero-based slot number.

There is one significant corner case in this scheme. When the contextCount is equal to a value in the contextArray, the oldest session needs to be recycled or flushed. In order to recycle the session, it must be loaded. To be loaded, there must be an available slot. Rather than require that a spare slot be available all the time, the TPM will check to see if the contextCount is equal to some value in the contextArray when a session is created. This prevents the last session slot from being used when it is likely that a session will need to be recycled.

If a TPM with both 1.2 and 2.0 functionality uses this scheme for both 1.2 and 2.0 sessions, and the list of active contexts is read with TPM_GetCapability(), the TPM will create 32-bit representations of the list that contains 16-bit values (the TPM2_GetCapability() returns a list of handles for active sessions rather than
a list of contextID). The full contextID has high-order bits that are either the same as the current contextCount or one less. It is one less if the 16-bits of the contextArray has a value that is larger than the low-order 16 bits of contextCount.

8.9.2 Includes, Defines, and Local Variables

```c
#define SESSION_C
#include "Tpm.h"
```

8.9.3 File Scope Function -- ContextIdSetOldest()

This function is called when the oldest contextID is being loaded or deleted. Once a saved context becomes the oldest, it stays the oldest until it is deleted.

Finding the oldest is a bit tricky. It is not just the numeric comparison of values but is dependent on the value of contextCounter.

Assume we have a small contextArray with 8, 4-bit values with values 1 and 2 used to indicate the loaded context slot number. Also assume that the array contains hex values of (0 0 1 0 3 0 9 F) and that the contextCounter is an 8-bit counter with a value of 0x37. Since the low nibble is 7, that means that values above 7 are older than values below it and, in this example, 9 is the oldest value.

Note if we subtract the counter value, from each slot that contains a saved contextID we get (- - - - B - 2 - 8) and the oldest entry is now easy to find.

```c
static void ContextIdSetOldest(
    void
) {
    CONTEXT_SLOT lowBits;
    CONTEXT_SLOT entry;
    CONTEXT_SLOT smallest = ((CONTEXT_SLOT)~0);
    UINT32 i;
    
    // Set oldestSaveContext to a value indicating none assigned
    s_oldestSavedSession = MAX_ACTIVE_SESSIONS + 1;
    
    lowBits = (CONTEXT_SLOT)gr.contextCounter;
    for(i = 0; i < MAX_ACTIVE_SESSIONS; i++) {
        entry = gr.contextArray[i];
        
        // only look at entries that are saved contexts
        if(entry > MAX_LOADED_SESSIONS) {
            // Use a less than or equal in case the oldest
            // is brand new (= lowBits-1) and equal to our initial
            // value for smallest.
            if(((CONTEXT_SLOT)(entry - lowBits)) <= smallest) {
                smallest = (entry - lowBits);
                s_oldestSavedSession = i;
            }
        }
    }
    
    // When we finish, either the s_oldestSavedSession still has its initial
    // value, or it has the index of the oldest saved context.
}
```
8.9.4 Startup Function -- SessionStartup()

This function initializes the session subsystem on TPM2_Startup().

```c
void SessionStartup(
    STARTUP_TYPE type
)
{
    UINT32 i;
    // Initialize session slots. At startup, all the in-memory session slots
    // are cleared and marked as not occupied
    for(i = 0; i < MAX_LOADED_SESSIONS; i++)
        s_sessions[i].occupied = FALSE; // session slot is not occupied
    // The free session slots the number of maximum allowed loaded sessions
    s_freeSessionSlots = MAX_LOADED_SESSIONS;
    // Initialize context ID data. On a ST_SAVE or hibernate sequence, it will
    // scan the saved array of session context counts, and clear any entry that
    // references a session that was in memory during the state save since that
    // memory was not preserved over the ST_SAVE.
    if(type == SU_RESUME || type == SU_RESTART)
    {
        // On ST_SAVE we preserve the contexts that were saved but not the ones
        // in memory
        for(i = 0; i < MAX_ACTIVE_SESSIONS; i++)
            { // If the array value is unused or references a loaded session then
                // that loaded session context is lost and the array entry is
                // reclaimed.
                if(gr.contextArray[i] <= MAX_LOADED_SESSIONS)
                    gr.contextArray[i] = 0;
            } // Find the oldest session in context ID data and set it in
    // s_oldestSavedSession
    ContextIdSetOldest();
    } else
    { // For STARTUP_CLEAR, clear out the contextArray
        for(i = 0; i < MAX_ACTIVE_SESSIONS; i++)
            gr.contextArray[i] = 0;
        // reset the context counter
        gr.contextCounter = MAX_LOADED_SESSIONS + 1;
        // Initialize oldest saved session
        s_oldestSavedSession = MAX_ACTIVE_SESSIONS + 1;
    } return;
}
```

8.9.5 Access Functions

8.9.5.1 SessionIsLoaded()

This function test a session handle references a loaded session. The handle must have previously been checked to make sure that it is a valid handle for an authorization session.
A PWAP authorization does not have a session.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>session is loaded</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>session is not loaded</td>
</tr>
</tbody>
</table>

### 8.9.5.2 SessionIsSaved()

This function tests a session handle references a saved session. The handle must have previously been checked to make sure that it is a valid handle for an authorization session.

NOTE: A password authorization does not have a session.

This function requires that the handle be a valid session handle.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>session is saved</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>session is not saved</td>
</tr>
</tbody>
</table>

```c
BOOL SessionIsSaved(
    TPM_HANDLE handle       // IN: session handle
) {
    pAssert(HandleGetType(handle) == TPM_HT_POLICY_SESSION
        || HandleGetType(handle) == TPM_HT_HMAC_SESSION);
    handle = handle & HR_HANDLE_MASK;
    // if out of range of possible active session, or not assigned, or
    // assigned to a loaded session, return false
    if(handle >= MAX_ACTIVE_SESSIONS
        || gr.contextArray[handle] == 0
        || gr.contextArray[handle] > MAX_LOADED_SESSIONS)
        return FALSE;
    return TRUE;
}
```
8.9.5.3 SequenceNumberForSavedContextIsValid()

This function validates that the sequence number and handle value within a saved context are valid.

```c
BOOL SequenceNumberForSavedContextIsValid(
    TPMS_CONTEXT *context // IN: pointer to a context structure to be validated
)
{
    #define MAX_CONTEXT_GAP ((UINT64)((CONTEXT_SLOT) ~0) + 1)
    TPM_HANDLE handle = context->savedHandle & HR_HANDLE_MASK;
    if(handle >= MAX_ACTIVE_SESSIONS // Handle must be with the range of active sessions
        || gr.contextArray[handle] <= MAX_LOADED_SESSIONS // the array entry must be for a saved context
        || gr.contextArray[handle] != (CONTEXT_SLOT)context->sequence // the array entry must agree with the sequence number
        || context->sequence > gr.contextCounter // the provided sequence number has to be less than the current counter
        || gr.contextCounter - context->sequence > MAX_CONTEXT_GAP)
        return FALSE;
    return TRUE;
}
```

8.9.5.4 SessionPCRValueIsCurrent()

This function is used to check if PCR values have been updated since the last time they were checked in a policy session.

This function requires the session is loaded.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>PCR value is current</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>PCR value is not current</td>
</tr>
</tbody>
</table>

```c
BOOL SessionPCRValueIsCurrent(
    SESSION *session  // IN: session structure
)
{
    if(session->pcrCounter != 0
        && session->pcrCounter != gr.pcrCounter
    )
        return FALSE;
    else
        return TRUE;
}
```

8.9.5.5 SessionGet()

This function returns a pointer to the session object associated with a session handle.

The function requires that the session is loaded.

```c
SESSION *
```
```c
SessionGet(
    TPM_HANDLE handle // IN: session handle
) {
    size_t slotIndex;
    CONTEXT_SLOT sessionIndex;
    pAssert(HandleGetType(handle) == TPM_HT_POLICY_SESSION
            || HandleGetType(handle) == TPM_HT_HMAC_SESSION);
    slotIndex = handle & HR_HANDLE_MASK;
    pAssert(slotIndex < MAX_ACTIVE_SESSIONS);
    // get the contents of the session array. Because session is loaded, we
    // should always get a valid sessionIndex
    sessionIndex = gr.contextArray[slotIndex] - 1;
    pAssert(sessionIndex < MAX_LOADED_SESSIONS);
    return &s_sessions[sessionIndex].session;
}
```

### 8.9.6 Utility Functions

#### 8.9.6.1 ContextIdSessionCreate()

This function is called when a session is created. It will check to see if the current gap would prevent a context from being saved. If so it will return TPM_RC_CONTEXT_GAP. Otherwise, it will try to find an open slot in `contextArray`, set `contextArray` to the slot.

This routine requires that the caller has determined the session array index for the session.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CONTEXT_GAP</td>
<td>can't assign a new <code>contextId</code> until the oldest saved session context is recycled</td>
</tr>
<tr>
<td>TPM_RC_SESSION_HANDLE</td>
<td>there is no slot available in the context array for tracking of this session context</td>
</tr>
</tbody>
</table>

```c
static TPM_RC ContextIdSessionCreate(
    TPM_HANDLE *handle, // OUT: receives the assigned handle. This will
    UINT32 sessionIndex // IN: The session context array entry that will
                     // be occupied by the created session
) {
    pAssert(sessionIndex < MAX_LOADED_SESSIONS);
    // check to see if creating the context is safe
    // Is this going to be an assignment for the last session context
    // array entry? If so, then there will be no room to recycle the
    // oldest context if needed. If the gap is not at maximum, then
    // it will be possible to save a context if it becomes necessary.
    if(s_oldestSavedSession < MAX_ACTIVE_SESSIONS
        && s_freeSessionSlots == 1)
    {
        // See if the gap is at maximum
```
205 // The current value of the contextCounter will be assigned to the next
206 // saved context. If the value to be assigned would make the same as an
207 // existing context, then we can’t use it because of the ambiguity it would
208 // create.
209 if ((CONTEXT_SLOT)gr.contextCounter
210 == gr.contextArray[s_oldestSavedSession])
211 return TPM_RC_CONTEXT_GAP;
212
213 // Find an unoccupied entry in the contextArray
214 for (*handle = 0; *handle < MAX_ACTIVE_SESSIONS; (*handle)++)
215 {
216 if (gr.contextArray[*handle] == 0)
217 {
218 // indicate that the session associated with this handle
219 // references a loaded session
220 gr.contextArray[*handle] = (CONTEXT_SLOT)(sessionIndex + 1);
221 return TPM_RC_SUCCESS;
222 }
223 }
224 return TPM_RC_SESSION_HANDLES;
225 }
226
8.9.6.2 SessionCreate()

This function does the detailed work for starting an authorization session. This is done in a support
routine rather than in the action code because the session management may differ in implementations.
This implementation uses a fixed memory allocation to hold sessions and a fixed allocation to hold the
contextID for the saved contexts.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CONTEXT_GAP</td>
<td>need to recycle sessions</td>
</tr>
<tr>
<td>TPM_RC_SESSION_HANDLE</td>
<td>active session space is full</td>
</tr>
<tr>
<td>TPM_RC_SESSION_MEMORY</td>
<td>loaded session space is full</td>
</tr>
</tbody>
</table>

227 TPM_RC
228 SessionCreate(
229 TPM_SE               sessionType, // IN: the session type
230 TPMI_ALG_HASH        authHash, // IN: the hash algorithm
231 TPM2B_NONCE          *nonceCaller, // IN: initial nonceCaller
232 TPM2B_DATA           *seed, // IN: seed data
233 TPM2B_DATA           *sessionHandle, // OUT: the session handle
234 TPM2B_NONCE          *nonceTpm, // OUT: the session nonce
235 )
236 {
237 TPM_RC               result = TPM_RC_SUCCESS;
238 CONTEXT_SLOT         slotIndex;
239 SESSION              *session = NULL;
240
241 pAssert(sessionType == TPM_SE_HMAC
242 || sessionType == TPM_SE_POLICY
243 || sessionType == TPM_SE_TRIAL);
244
245 // If there are no open spots in the session array, then no point in searching
246 if (s_freeSessionSlots == 0)
247 return TPM_RC_SESSION_MEMORY;
248
249 // Find a space for loading a session
250 for(slotIndex = 0; slotIndex < MAX_LOADED_SESSIONS; slotIndex++)
253  {
254      // Is this available?
255      if (s_sessions[slotIndex].occupied == FALSE) {
256          session = &s_sessions[slotIndex].session;
257          break;
258      }
259  }
260  // if no spot found, then this is an internal error
261  if (slotIndex >= MAX_LOADED_SESSIONS)
262      FAIL(FATAL_ERROR_INTERNAL);
263  // Call context ID function to get a handle. TPM_RC_SESSION_HANDLE may be
264  // returned from ContextIdHandleAssign()
265  result = ContextIdSessionCreate(sessionHandle, slotIndex);
266  if (result != TPM_RC_SUCCESS)
267      return result;
268  //*** Only return from this point on is TPM_RC_SUCCESS
269  // Can now indicate that the session array entry is occupied.
270  s_freeSessionSlots--;
271  s_sessions[slotIndex].occupied = TRUE;
272  // Initialize the session data
273  MemorySet(session, 0, sizeof(SESSION));
274  // Initialize internal session data
275  session->authHashAlg = authHash;
276  // Initialize session type
277  if (sessionType == TPM_SE_HMAC) {
278      *sessionHandle += HMAC_SESSION_FIRST;
279  } else {
280      *sessionHandle += POLICY_SESSION_FIRST;
281  }
282  // For TPM_SE_POLICY or TPM_SE_TRIAL
283  session->attributes.isPolicy = SET;
284  if (sessionType == TPM_SE_TRIAL)
285      session->attributes.isTrialPolicy = SET;
286  SessionSetStartTime(session);
287  // Initialize policyDigest. policyDigest is initialized with a string of 0
288  // of session algorithm digest size. Since the session is already clear.
289  // Just need to set the size
290  session->u2.policyDigest.t.size =
291      CryptHashGetDigestSize(session->authHashAlg);
292  // Create initial session nonce
293  session->nonceTPM.t.size = nonceCaller->t.size;
294  CryptRandomGenerate(session->nonceTPM.t.size, session->nonceTPM.t.buffer);
295  MemoryCopy2B(&nonceTpm->b, &session->nonceTPM.b,
296      sizeof(nonceTpm->t.buffer));
297  // Set up session parameter encryption algorithm
298  session->symmetric = *symmetric;
299  // If there is a bind object or a session secret, then need to compute
300  // a sessionKey.
301  if (bind != TPM_RH_NULL || seed->t.size != 0) {
302      // sessionKey = KDFa(hash, (authValue || seed), "ATH", nonceTPM,
303          nonceCaller, bits)
The HMAC key for generating the sessionSecret can be the concatenation of an authorization value and a seed value:

```c
TPM2B_TYPE (KEY, (sizeof (TPMT_HA) + sizeof (seed->t.buffer)));
TPM2B_KEY key;
```

Get hash size, which is also the length of sessionKey:

```c
session->sessionKey.t.size = CryptHashGetDigestSize (session->authHashAlg);
```

Get authValue of associated entity:

```c
EntityGetAuthValue (bind, (TPM2B_AUTH *)&key);
pAssert (key.t.size + seed->t.size <= sizeof (key.t.buffer));
```

Concatenate authValue and seed:

```c
MemoryConcat2B (&key.b, &seed->b, sizeof (key.t.buffer));
```

Compute the session key:

```c
CryptKDFa (session->authHashAlg, &key.b, SESSION_KEY, &session->nonceTPM.b, 
&nonceCaller->b, 
session->sessionKey.t.size * 8, session->sessionKey.t.buffer, 
NULL, FALSE);
```

Copy the name of the entity that the HMAC session is bound to:

```c
// Policy session is not bound to an entity
if (bind != TPM_RH_NULL && sessionType == TPM_SE_HMAC) 
{
    session->attributes.isBound = SET;
    SessionComputeBoundEntity (bind, &session->u1.boundEntity);
}
```

If there is a bind object and it is subject to DA, then use of this session is subject to DA regardless of how it is used:

```c
if (IsDAExempted (bind) == FALSE);
```

If the session is bound, then check to see if it is bound to lockoutAuth:

```c
if (bind == TPM_RH_LOCKOUT);
return TPM_RC_SUCCESS;
```

### 8.9.6.3 SessionContextSave()

This function is called when a session context is to be saved. The contextID of the saved session is returned. If no contextID can be assigned, then the routine returns TPM_RC_CONTEXT_GAP. If the function completes normally, the session slot will be freed.

This function requires that `handle` references a loaded session. Otherwise, it should not be called at the first place.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CONTEXT_GAP</td>
<td>A contextID could not be assigned.</td>
</tr>
<tr>
<td>TPM_RC_TOO_MANY_CONTEXTS</td>
<td>The contextID counter maxed out</td>
</tr>
</tbody>
</table>

```c
TPM_RC SessionContextSave(
TPM_HANDLE handle,       // IN: session handle
CONTEXT_COUNTER *contextID // OUT: assigned contextID
)
{
UINT32 contextIndex;
CONTEXT_SLOT slotIndex;
pAssert (SessionIsLoaded (handle));
```
8.9.6.4 SessionContextLoad()

This function is used to load a session from saved context. The session handle must be for a saved context.

If the gap is at a maximum, then the only session that can be loaded is the oldest session, otherwise TPM_RC_CONTEXT_GAP is returned.

This function requires that handle references a valid saved session.
Table: Error Returns

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SESSION_MEMORY</td>
<td>no free session slots</td>
</tr>
<tr>
<td>TPM_RC_CONTEXT_GAP</td>
<td>the gap count is maximum and this is not the oldest saved context</td>
</tr>
</tbody>
</table>

```c
TPM_RC
SessionContextLoad(
    SESSION_BUF *session, // IN: session structure from saved context
    TPM_HANDLE *handle    // IN/OUT: session handle
)
{
    UINT32    contextIndex;
    CONTEXT_SLOT slotIndex;
    pAssert(HandleGetType(*handle) == TPM_HT_POLICY_SESSION
        || HandleGetType(*handle) == TPM_HT_HMAC_SESSION);
    // Don't bother looking if no openings
    if(s_freeSessionSlots == 0)
        return TPM_RC_SESSION_MEMORY;
    // Find a free session slot to load the session
    for(slotIndex = 0; slotIndex < MAX_LOADED_SESSIONS; slotIndex++)
        if(s_sessions[slotIndex].occupied == FALSE) break;
    // if no spot found, then this is an internal error
    pAssert(slotIndex < MAX_LOADED_SESSIONS);
    contextIndex = *handle & HR_HANDLE_MASK; // extract the index
    // If there is only one slot left, and the gap is at maximum, the only session
    // context that we can safely load is the oldest one.
    if(s_oldestSavedSession < MAX_ACTIVE_SESSIONS
        && s_freeSessionSlots == 1
        && (CONTEXT_SLOT)gr.contextCounter == gr.contextArray[s_oldestSavedSession]
        && contextIndex != s_oldestSavedSession)
        return TPM_RC_CONTEXT_GAP;
    pAssert(contextIndex < MAX_ACTIVE_SESSIONS);
    // set the contextArray value to point to the session slot where
    // the context is loaded
    gr.contextArray[contextIndex] = slotIndex + 1;
    // if this was the oldest context, find the new oldest
    if(contextIndex == s_oldestSavedSession)
        ContextIdSetOldest();
    // Copy session data to session slot
    MemoryCopy(&s_sessions[slotIndex].session, session, sizeof(SESSION));
    // Set session slot as occupied
    s_sessions[slotIndex].occupied = TRUE;
    // Reduce the number of open spots
    s_freeSessionSlots--;
    return TPM_RC_SUCCESS;
}
```
8.9.6.5  SessionFlush()

This function is used to flush a session referenced by its handle. If the session associated with handle is loaded, the session array entry is marked as available.

This function requires that handle be a valid active session.

```c
void SessionFlush(
    TPM_HANDLE handle     // IN: loaded or saved session handle
{
    CONTEXT_SLOT slotIndex;
    UINT32    contextIndex;  // Index into contextArray
    pAssert((HandleGetType(handle) == TPM_HT_POLICY_SESSION
            || HandleGetType(handle) == TPM_HT_HMAC_SESSION
            )
            && (SessionIsLoaded(handle) || SessionIsSaved(handle))
    );
    // Flush context ID of this session
    // Convert handle to an index into the contextArray
    contextIndex = handle & HR_HANDLE_MASK;
    pAssert(contextIndex < sizeof(gr.contextArray) / sizeof(gr.contextArray[0]));
    // Get the current contents of the array
    slotIndex = gr.contextArray[contextIndex];
    // Mark context array entry as available
    gr.contextArray[contextIndex] = 0;
    // Is this a saved session being flushed
    if(slotIndex > MAX_LOADED_SESSIONS)
    {
        // Flushing the oldest session?
        if(contextIndex == s_oldestSavedSession)
            // If so, find a new value for oldest.
            ContextIdSetOldest();
    }
    else
    {
        // Adjust slot index to point to session array index
        slotIndex -= 1;
        // Free session array index
        s_sessions[slotIndex].occupied = FALSE;
        s_freeSessionSlots++;
    }
    return;
}
```

8.9.6.6  SessionComputeBoundEntity()

This function computes the binding value for a session. The binding value for a reserved handle is the handle itself. For all the other entities, the authValue at the time of binding is included to prevent squatting. For those values, the Name and the authValue are concatenated into the bind buffer. If they will not both fit, the will be overlapped by XORing() bytes. If XOR is required, the bind value will be full.

```c
void SessionComputeBoundEntity(
```
TPMI_DH_ENTITY entityHandle, // IN: handle of entity
TPM2B_NAME *bind // OUT: binding value
}
{
  TPM2B_AUTH auth;
  BYTE *pAuth = auth.t.buffer;
  UINT16 i;

  // Get name
  EntityGetName(entityHandle, bind);
  // The bound value of a reserved handle is the handle itself
  if(bind->t.size == sizeof(TPM_HANDLE)) return;
  // For all the other entities, concatenate the authorization value to the name.
  // Get a local copy of the authorization value because some overlapping
  // may be necessary.
  EntityGetAuthValue(entityHandle, &auth);
  // Make sure that the extra space is zeroed
  MemorySet(&bind->t.name[bind->t.size], 0, sizeof(bind->t.name) - bind->t.size);
  // XOR the authValue at the end of the name
  for(i = sizeof(bind->t.name) - auth.t.size; i < sizeof(bind->t.name); i++)
    bind->t.name[i] ^= *pAuth++;

  // Set the bind value to the maximum size
  bind->t.size = sizeof(bind->t.name);
  return;
}

8.9.6.7 SessionSetStartTime()

This function is used to initialize the session timing.

void SessionSetStartTime(
  SESSION *session // IN: the session to update
)
{
  session->startTime = g_time;
  session->epoch = g_timeEpoch;
  session->timeout = 0;
}

8.9.6.8 SessionResetPolicyData()

This function is used to reset the policy data without changing the nonce or the start time of the session.

void SessionResetPolicyData(
  SESSION *session // IN: the session to reset
)
{
  SESSION_ATTRIBUTES oldAttributes;
pAssert(session != NULL);
  // Will need later
  oldAttributes = session->attributes;
  // No command
  session->commandCode = 0;
MemorySet(&session->commandLocality, 0, sizeof(session->commandLocality));

// The cpHash size to zero
session->u1.cpHash.b.size = 0;

// No timeout
session->timeout = 0;

// Reset the pcrCounter
session->pcrCounter = 0;

// Reset the policy hash
MemorySet(&session->u2.policyDigest.t.buffer, 0,
          sizeof(session->u2.policyDigest.t.size));

// Reset the session attributes
MemorySet(&session->attributes, 0, sizeof(SESSION_ATTRIBUTES));

// Restore the policy attributes
session->attributes.isPolicy = SET;
session->attributes.isTrialPolicy = oldAttributes.isTrialPolicy;

// Restore the bind attributes
session->attributes.isDaBound = oldAttributes.isDaBound;
session->attributes.isLockoutBound = oldAttributes.isLockoutBound;

8.9.6.9 SessionCapGetLoaded()

This function returns a list of handles of loaded session, started from input handle.

Handle must be in valid loaded session handle range, but does not have to point to a loaded session.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more handles available</td>
</tr>
<tr>
<td>NO</td>
<td>all the available handles has been returned</td>
</tr>
</tbody>
</table>

TPMI_YES_NO

SessionCapGetLoaded(
    TPMI_SH_POLICY   handle,  // IN: start handle
    UINT32           count,   // IN: count of returned handles
    TPML_HANDLE     *handleList // OUT: list of handle
)
{
    TPMI_YES_NO     more = NO;
    UINT32          i;
    pAssert(HandleGetType(handle) == TPM_HT_LOADED_SESSION);

    // Initialize output handle list
    handleList->count = 0;

    // The maximum count of handles we may return is MAX_CAP_HANDLES
    if(count > MAX_CAP_HANDLES) count = MAX_CAP_HANDLES;

    // Iterate session context ID slots to get loaded session handles
    for(i = handle & HR_HANDLE_MASK; i < MAX_ACTIVE_SESSIONS; i++)
    {
        // If session is active
        if(gr.contextArray[i] != 0)
        {
            // If session is loaded

            MemorySet(&session->commandLocality, 0, sizeof(session->commandLocality));

            // The cpHash size to zero
            session->u1.cpHash.b.size = 0;

            // No timeout
            session->timeout = 0;

            // Reset the pcrCounter
            session->pcrCounter = 0;

            // Reset the policy hash
            MemorySet(&session->u2.policyDigest.t.buffer, 0,
                      sizeof(session->u2.policyDigest.t.size));

            // Reset the session attributes
            MemorySet(&session->attributes, 0, sizeof(SESSION_ATTRIBUTES));

            // Restore the policy attributes
            session->attributes.isPolicy = SET;
            session->attributes.isTrialPolicy = oldAttributes.isTrialPolicy;

            // Restore the bind attributes
            session->attributes.isDaBound = oldAttributes.isDaBound;
            session->attributes.isLockoutBound = oldAttributes.isLockoutBound;
        }
    }
if (gr.contextArray[i] <= MAX_LOADED_SESSIONS)
{
    if (handleList->count < count)
    {
        SESSION *session;

        // If we have not filled up the return list, add this
        // session handle to it
        // assume that this is going to be an HMAC session
        handle = i + HMAC_SESSION_FIRST;
        session = SessionGet(handle);
        if (session->attributes.isPolicy)
            handle = i + POLICY_SESSION_FIRST;
        handleList->handle[handleList->count] = handle;
        handleList->count++;
    }
}
else
{
    // If the return list is full but we still have loaded object
    // available, report this and stop iterating
    more = YES;
    break;
}
}
return more;

8.9.6.10 SessionCapGetSaved()

This function returns a list of handles for saved session, starting at handle. Handle must be in a valid handle range, but does not have to point to a saved session.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more handles available</td>
</tr>
<tr>
<td>NO</td>
<td>all the available handles has been returned</td>
</tr>
</tbody>
</table>

TPMI_YES_NO SessionCapGetSaved(
    TPMI_SH_HMAC handle,       // IN: start handle
    UINT32 count,              // IN: count of returned handles
    TPML_HANDLE *handleList,   // OUT: list of handle
)

#define TPM_HT_SAVED_SESSION
pAssert(HandleGetType(handle) == TPM_HT_SAVED_SESSION);
#else
pAssert(HandleGetType(handle) == TPM_HT_ACTIVE_SESSION);
#endif

// Initialize output handle list
handleList->count = 0;

// The maximum count of handles we may return is MAX_CAP_HANDLES
if (count > MAX_CAP_HANDLES) count = MAX_CAP_HANDLES;

// Iterate session context ID slots to get loaded session handles
for(i = handle & HR_HANDLE_MASK; i < MAX_ACTIVE_SESSIONS; i++)
{
    // If session is active
    if(gr.contextArray[i] != 0)
    {
        // If session is saved
        if(gr.contextArray[i] > MAX_LOADED_SESSIONS)
        {
            if(handleList->count < count)
            {
                // If we have not filled up the return list, add this
                // session handle to it
                handleList->handle[handleList->count] = i + HMAC_SESSION_FIRST;
                handleList->count++;
            }
            else
            {
                // If the return list is full but we still have loaded object
                // available, report this and stop iterating
                more = YES;
                break;
            }
        }
        else
        {
            // If session is active
            // But we have not saved it
            // If we have not filled up the return list, add this
            // session handle to it
            handleList->handle[handleList->count] = i + HMAC_SESSION_FIRST;
            handleList->count++;
        }
    }
}

return more;

8.9.6.11 SessionCapGetLoadedNumber()

This function return the number of authorization sessions currently loaded into TPM RAM.

UINT32 SessionCapGetLoadedNumber()
{
    return MAX_LOADED_SESSIONS - _freeSessionSlots;
}

8.9.6.12 SessionCapGetLoadedAvail()  

This function returns the number of additional authorization sessions, of any type, that could be loaded into TPM RAM.

NOTE: In other implementations, this number may just be an estimate. The only requirement for the estimate is, if it is one or more, then at least one session must be loadable.

UINT32 SessionCapGetLoadedAvail()
{
    return _freeSessionSlots;
}

8.9.6.13 SessionCapGetActiveNumber()

This function returns the number of active authorization sessions currently being tracked by the TPM.
8.9.6.14 SessionCapGetActiveAvail()

This function returns the number of additional authorization sessions, of any type, that could be created. This not the number of slots for sessions, but the number of additional sessions that the TPM is capable of tracking.

```c
UINT32 SessionCapGetActiveAvail(
    void
) {
    UINT32 i;
    UINT32 num = 0;
    for(i = 0; i < MAX_ACTIVE_SESSIONS; i++) {
        if(gr.contextArray[i] == 0) num++;
    }
    return num;
}
```
8.10 Time.c

8.10.1 Introduction

This file contains the functions relating to the TPM's time functions including the interface to the implementation-specific time functions.

8.10.2 Includes

```c
#include "Tpm.h"
#include "PlatformData.h"
```

8.10.3 Functions

8.10.3.1 TimePowerOn()

This function initialize time info at _TPM_Init().

This function is called at _TPM_Init() so that the TPM time can start counting as soon as the TPM comes out of reset and doesn't have to wait until TPM2_Startup() in order to begin the new time epoch. This could be significant for systems that could get powered up but not run any TPM commands for some period of time.

```c
void
TimePowerOn(
    void
) {
    g_time = _plat__TimerRead();
}
```

8.10.3.2 TimeNewEpoch()

This function does the processing to generate a new time epoch nonce and set NV for update. This function is only called when NV is known to be available and the clock is running. The epoch is updated to persistent data.

```c
static void
TimeNewEpoch(
    void
) {
    #if CLOCK_STOPS
        CryptRandomGenerate(sizeof(CLOCK_NONCE), (BYTE *)&g_timeEpoch);
    #else
        // if the epoch is kept in NV, update it.
        gp.timeEpoch++;
        NV_SYNC_PERSISTENT(timeEpoch);
    #endif
    // Clean out any lingering state
    _plat__TimerWasStopped();
}
```

8.10.3.3 TimeStartup()

This function updates the resetCount and restartCount components of TPMS_CLOCK_INFO structure at TPM2_Startup().
This function will deal with the deferred creation of a new epoch. TimeUpdateToCurrent() will not start a new epoch even if one is due when TPM_Startup() has not been run. This is because the state of NV is not known until startup completes. When Startup is done, then it will create the epoch nonce to complete the initializations by calling this function.

```c
void
Tim

eStartup
(STARTUP_TYPE     type     // IN: start up type)
{}
NOT_REFERENCED(type);
// If the previous cycle is orderly shut down, the value of the safe bit
// the same as previously saved. Otherwise, it is not safe.
if(!NV_IS_ORDERLY)
go.clockSafe = NO;
return;
}
```

### 8.10.3.4 TimeClockUpdate()

This function updates go.clock. If newTime requires an update of NV, then NV is checked for availability. If it is not available or is rate limiting, then go.clock is not updated and the function returns an error. If newTime would not cause an NV write, then go.clock is updated. If an NV write occurs, then go.safe is SET.

```c
void
Tim
leClockUpdate(UINT64           newTime   // IN: New time value in mS.
){
#define CLOCK_UPDATE_MASK  ((1ULL << NV_CLOCK_UPDATE_INTERVAL) - 1)
// Check to see if the update will cause a need for an nvClock update
if((newTime | CLOCK_UPDATE_MASK) > (go.clock | CLOCK_UPDATE_MASK))
{
pAssert(g_NvStatus == TPM_RC_SUCCESS);
// Going to update the NV time state so SET the safe flag
go.clockSafe = YES;
// update the time
go.clock = newTime;
NvWrite(NV_ORDERLY_DATA, sizeof(go), &go);
}
else
// No NV update needed so just update
go.clock = newTime;
}
```

### 8.10.3.5 TimeUpdate()

This function is used to update the time and clock values. If the TPM has run TPM2_Startup(), this function is called at the start of each command. If the TPM has not run TPM2_Startup(), this is called from TPM2_Startup() to get the clock values initialized. It is not called on command entry because, in this implementation, the go structure is not read from NV until TPM2_Startup(). The reason for this is that the initialization code (_TPM_Init()) may run before NV is accessible.

```c
void
Tim

Update()
```
8.10.3.6 TimeUpdateToCurrent()

This function updates the Time and Clock in the global TPMS_TIME_INFO structure.

In this implementation, Time and Clock are updated at the beginning of each command and the values are unchanged for the duration of the command.

Because Clock updates may require a write to NV memory, Time and Clock are not allowed to advance if NV is not available. When clock is not advancing, any function that uses Clock will fail and return TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE.

This implementation does not do rate limiting. If the implementation does do rate limiting, then the Clock update should not be inhibited even when doing rate limiting.

```c
87 void
88 TimeUpdateToCurrent(
89     void
90 )
91 {
92     // Can't update time during the dark interval or when rate limiting so don't
93     // make any modifications to the internal clock value. Also, defer any clock
94     // processing until TPM has run TPM2_Startup()
95     if (!NV_IS_AVAILABLE || !TPMIsStarted())
96         return;
97     TimeUpdate();
98 }
```

8.10.3.7 TimeSetAdjustRate()

This function is used to perform rate adjustment on Time and Clock.

```c
100 void
101 TimeSetAdjustRate(
102     TPM_CLOCK_ADJUST adjust     // IN: adjust constant
103 )
104 {
105     switch (adjust)
106     {
```
8.10.3.8 TimeGetMarshaled()

This function is used to access TPMS_TIME_INFO in canonical form. The function collects the time information and marshals it into dataBuffer and returns the marshaled size.

```c
UINT16
TimeGetMarshaled(
  TIME_INFO  *dataBuffer  // OUT: result buffer
)
{
  TPMS_TIME_INFO  timeInfo;
  // Fill TPMS_TIME_INFO structure
  timeInfo.time = g_time;
  TimeFillInfo(&timeInfo.clockInfo);
  // Marshal TPMS_TIME_INFO to canonical form
  return TPMS_TIME_INFO_Marshal(&timeInfo, (BYTE **)&dataBuffer, NULL);
}
```

8.10.3.9 TimeFillInfo

This function gathers information to fill in a TPMS_CLOCK_INFO structure.

```c
void
TimeFillInfo(
  TPMS_CLOCK_INFO  *clockInfo
)
{
  clockInfo->clock = go.clock;
  clockInfo->resetCount = gp.resetCount;
  clockInfo->restartCount = gr.restartCount;
  // If NV is not available, clock stopped advancing and the value reported is
  // not "safe".
```

if(NV_IS_AVAILABLE)
    clockInfo->safe = go.clockSafe;
else
    clockInfo->safe = NO;

return;
9 Support

9.1 AlgorithmCap.c

9.1.1 Description

This file contains the algorithm property definitions for the algorithms and the code for the TPM2_GetCapability() to return the algorithm properties.

9.1.2 Includes and Defines

```c
#include "Tpm.h"
typedef struct
{
    TPM_ALG_ID          algID;
    TPMA_ALGORITHM      attributes;
} ALGORITHM;
static const ALGORITHM    s_algorithms[] =
{
    // The entries in this table need to be in ascending order but the table doesn't
    // need to be full (gaps are allowed). One day, a tool might exist to fill in the
    // table from the TPM_ALG description
    #if ALG_RSA
    {TPM_ALG_RSA,           TPMA_ALGORITHM_INITIALIZER(1, 0, 1, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_TDES
    {TPM_ALG_TDES,          TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_SHA1
    {TPM_ALG_SHA1,          TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_HMAC
    {TPM_ALG_HMAC,          TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 1, 0, 0, 0)},
    #endif
    #if ALG_AES
    {TPM_ALG_AES,           TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_MGF1
    {TPM_ALG_MGF1,          TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 1, 0)},
    #endif
    #if ALG_KEYEDHASH
    {TPM_ALG_KEYEDHASH,     TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 1, 0, 1, 1, 0, 0)},
    #endif
    #if ALG_XOR
    {TPM_ALG_XOR,           TPMA_ALGORITHM_INITIALIZER(0, 1, 1, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_SHA256
    {TPM_ALG_SHA256,        TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_SHA384
    {TPM_ALG_SHA384,        TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_SHA512
    {TPM_ALG_SHA512,        TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_SM3_256
    {TPM_ALG_SM3_256,       TPMA_ALGORITHM_INITIALIZER(0, 0, 1, 0, 0, 0, 0, 0, 0)},
    #endif
    #if ALG_SM4
    {TPM_ALG_SM4,           TPMA_ALGORITHM_INITIALIZER(0, 1, 0, 0, 0, 0, 0, 0, 0)},
    #endif
```
#if ALG_RSASSA
    {TPM_ALG_RSASSA, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 1, 0, 0, 0, 0)},
#endif

#if ALG_RSAS
    {TPM_ALG_RSAS, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 1, 0, 0, 0, 0)},
#endif

#if ALG_RSA
    {TPM_ALG_RSA, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 1, 0, 0, 0, 0)},
#endif

#if ALG_RSAPSS
    {TPM_ALG_RSAPSS, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif

#if ALG_OAEP
    {TPM_ALG_OAEP, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif

#if ALG_ECDSA
    {TPM_ALG_ECDSA, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 1, 0)},
#endif

#if ALG_ECDH
    {TPM_ALG_ECDH, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif

#if ALG_ECDAA
    {TPM_ALG_ECDAA, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif

#if ALG_SM2
    {TPM_ALG_SM2, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 1, 0, 1, 0)},
#endif

#if ALG_ECSCHNORR
    {TPM_ALG_ECSCHNORR, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif

#if ALG_ECMQV
    {TPM_ALG_ECMQV, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif

#if ALG_KDF1_SP800_56A
    {TPM_ALG_KDF1_SP800_56A, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 1, 0, 0, 0, 0, 0)},
#endif

#if ALG_KDF2
    {TPM_ALG_KDF2, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 1, 0, 0, 0, 0, 0)},
#endif

#if ALG_KDF1_SP800_108
    {TPM_ALG_KDF1_SP800_108, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 1, 0, 0, 0, 0, 0)},
#endif

#if ALG_ECC
    {TPM_ALG_ECC, TPMA_ALGORITHM_INITIALIZER(1, 0, 0, 1, 0, 0, 0, 0, 0)},
#endif

#if ALG_SYMCIPHER
    {TPM_ALG_SYMCIPHER, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 0, 1, 0, 0, 0, 0)},
#endif

#if ALG_CAMELLIA
    {TPM_ALG_CAMELLIA, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 0, 0, 1, 0, 0, 0)},
#endif

#if ALG_CMAC
    {TPM_ALG_CMAC, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif

#if ALG_CTR
    {TPM_ALG_CTR, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif

#if ALG_OFB
    {TPM_ALG_OFB, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif

#if ALG_CBC
    {TPM_ALG_CBC, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif

#if ALG_CFB
    {TPM_ALG_CFB, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif

#if ALG_ECB
    {TPM_ALG_ECB, TPMA_ALGORITHM_INITIALIZER(0, 0, 0, 0, 0, 0, 1, 0, 0)},
#endif
9.1.3 AlgorithmCapGetImplemented()

This function is used by TPM2_GetCapability() to return a list of the implemented algorithms.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>more algorithms to report</td>
</tr>
<tr>
<td>NO</td>
<td>no more algorithms to report</td>
</tr>
</tbody>
</table>

```c
TPMI_YES_NO
AlgorithmCapGetImplemented(
    TPM_ALG_ID algID,       // IN: the starting algorithm ID
    UINT32 count,           // IN: count of returned algorithms
    TPM_ALG_PROPERTY *algList // OUT: algorithm list
) {
    TPMI_YES_NO more = NO;
    UINT32 i;
    UINT32 algNum;
    // initialize output algorithm list
    algList->count = 0;
    // The maximum count of algorithms we may return is MAX_CAP_ALGS.
    if(count > MAX_CAP_ALGS)
        count = MAX_CAP_ALGS;
    // Compute how many algorithms are defined in s_algorithms array.
    algNum = sizeof(s_algorithms) / sizeof(s_algorithms[0]);
    // Scan the implemented algorithm list to see if there is a match to 'algID'.
    for(i = 0; i < algNum; i++)
        if(s_algorithms[i].algID < algID)
            continue;
        if(algList->count < count)
            { // If algID is less than the starting algorithm ID, skip it
                if(s_algorithms[i].algID < algID)
                    continue;
                if(algList->count < count)
                    { // If we have not filled up the return list, add more algorithms
                        algList->algProperties[algList->count].alg = s_algorithms[i].algID;
                        algList->algProperties[algList->count].algProperties =
                            s_algorithms[i].attributes;
                        algList->count++;
                    }
            } else
                { // If the return list is full but we still have algorithms
                    more = YES;
                    break;
                }
    return more;
}
```

9.1.4 AlgorithmGetImplementedVector()

This function returns the bit vector of the implemented algorithms.
LIB_EXPORT
void
AlgorithmGetImplementedVector(
    ALGORITHM_VECTOR  *implemented  // OUT: the implemented bits are SET
)
{
    int            index;

    // Nothing implemented until we say it is
    MemorySet(implemented, 0, sizeof(ALGORITHM_VECTOR));

    for(index = (sizeof(s_algorithms) / sizeof(s_algorithms[0])) - 1;
        index >= 0;
        index--)
        SET_BIT(s_algorithms[index].algID, *implemented);

    return;
}
9.2 Bits.c

9.2.1 Introduction

This file contains bit manipulation routines. They operate on bit arrays. The 0th bit in the array is the right-most bit in the 0th octet in the array.

NOTE: If pAssert() is defined, the functions will assert if the indicated bit number is outside of the range of bArray. How the assert is handled is implementation dependent.

9.2.2 Includes

```
#include "Tpm.h"
```

9.2.3 Functions

9.2.3.1 TestBit()

This function is used to check the setting of a bit in an array of bits.

```
BOOL TestBit(
    unsigned int bitNum, // IN: number of the bit in 'bArray'
    BYTE *bArray,       // IN: array containing the bits
    unsigned int bytesInArray // IN: size in bytes of 'bArray'
)
{
    pAssert(bytesInArray > (bitNum >> 3));
    return ((bArray[bitNum >> 3] & (1 << (bitNum & 7))) != 0);
}
```

9.2.3.2 SetBit()

This function will set the indicated bit in bArray.

```
void SetBit(
    unsigned int bitNum, // IN: number of the bit in 'bArray'
    BYTE *bArray,       // IN: array containing the bits
    unsigned int bytesInArray // IN: size in bytes of 'bArray'
)
{
    pAssert(bytesInArray > (bitNum >> 3));
    bArray[bitNum >> 3] |= (1 << (bitNum & 7));
}
```

9.2.3.3 ClearBit()

This function will clear the indicated bit in bArray.

```
void ClearBit(
    unsigned int bitNum, // IN: number of the bit in 'bArray'
    BYTE *bArray,       // IN: array containing the bits
    unsigned int bytesInArray // IN: size in bytes of 'bArray'
)
{
    pAssert(bytesInArray > (bitNum >> 3));
    bArray[bitNum >> 3] &= ~(1 << (bitNum & 7));
}
```
ClearBit(
    unsigned int bitNum, // IN: number of the bit in 'bArray'.
    BYTE *bArray,      // IN: array containing the bits
    unsigned int bytesInArray // IN: size in bytes of 'bArray'
)
{
    pAssert(bytesInArray > (bitNum >> 3));
    bArray[bitNum >> 3] &= ~(1 << (bitNum & 7));
}
9.3  CommandCodeAttributes.c

9.3.1  Introduction

This file contains the functions for testing various command properties.

9.3.2  Includes and Defines

#include "Tpm.h"
#include "CommandCodeAttributes_fp.h"

Set the default value for CC_VEND if not already set

#ifndef CC_VEND
#define CC_VEND (TPM_CC)(0x20000000)
#endif

typedef UINT16 ATTRIBUTE_TYPE;

The following file is produced from the command tables in part 3 of the specification. It defines the attributes for each of the commands.

NOTE: This file is currently produced by an automated process. Files produced from Part 2 or Part 3 tables through automated processes are not included in the specification so that there is no ambiguity about the table containing the information being the normative definition.

#define _COMMAND_CODE_ATTRIBUTES_
#include "CommandAttributeData.h"

9.3.3  Command Attribute Functions

9.3.3.1  NextImplementedIndex()

This function is used when the lists are not compressed. In a compressed list, only the implemented commands are present. So, a search might find a value but that value may not be implemented. This function checks to see if the input commandIndex points to an implemented command and, if not, it searches upwards until it finds one. When the list is compressed, this function gets defined as a no-op.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIMPLEMENTED_COMMAND_INDEX</td>
<td>command is not implemented</td>
</tr>
<tr>
<td>other</td>
<td>index of the command</td>
</tr>
</tbody>
</table>

#if !COMPRESSED_LISTS
static COMMAND_INDEX NextImplementedIndex(
    COMMAND_INDEX commandIndex
)
{
    for(;commandIndex < COMMAND_COUNT; commandIndex++)
    {
        if(s_commandAttributes[commandIndex] & IS_IMPLEMENTED)
            return commandIndex;
    }
    return UNIMPLEMENTED_COMMAND_INDEX;
}
#else
#define NextImplementedIndex(x) (x)
#endif
9.3.3.2 GetClosestCommandIndex()

This function returns the command index for the command with a value that is equal to or greater than the input value.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIMPLEMENTED_COMMAND_INDEX</td>
<td>command is not implemented</td>
</tr>
<tr>
<td>other</td>
<td>index of a command</td>
</tr>
</tbody>
</table>

```
COMMAND_INDEX GetClosestCommandIndex(
    TPM_CC commandCode      // IN: the command code to start at
) {
    BOOL vendor = (commandCode & CC_VEND) != 0;
    COMMAND_INDEX searchIndex = (COMMAND_INDEX)commandCode;

    // The commandCode is a UINT32 and the search index is UINT16. We are going to
    // search for a match but need to make sure that the commandCode value is not
    // out of range. To do this, need to clear the vendor bit of the commandCode
    // (if set) and compare the result to the 16-bit searchIndex value. If it is
    // out of range, indicate that the command is not implemented
    if((commandCode & ~CC_VEND) != searchIndex)
        return UNIMPLEMENTED_COMMAND_INDEX;

    // if there is at least one vendor command, the last entry in the array will
    // have the v bit set. If the input commandCode is larger than the last
    // vendor-command, then it is out of range.
    if(vendor)
    {
        #if VENDOR_COMMAND_ARRAY_SIZE > 0
            COMMAND_INDEX commandIndex;
            COMMAND_INDEX min;
            COMMAND_INDEX max;
            int diff;
        #if LIBRARY_COMMAND_ARRAY_SIZE == COMMAND_COUNT
            #error "Constants are not consistent."
        #endif

            // Check to see if the value is equal to or below the minimum
            // entry.
            // Note: Put this check first so that the typical case of only one vendor-
            // specific command doesn't waste any more time.
            if(GET_ATTRIBUTE(s_ccAttr[LIBRARY_COMMAND_ARRAY_SIZE], TPMA_CC,
                commandIndex) >= searchIndex)
            {
                // the vendor array is always assumed to be packed so there is
                // no need to check to see if the command is implemented
                return LIBRARY_COMMAND_ARRAY_SIZE;
            }

            // See if this is out of range on the top
            if(GET_ATTRIBUTE(s_ccAttr[COMMAND_COUNT - 1], TPMA_CC, commandIndex) < searchIndex)
            {
                return UNIMPLEMENTED_COMMAND_INDEX;
            }

            commandIndex = UNIMPLEMENTED_COMMAND_INDEX; // Needs initialization to keep
            // compiler happy
            min = LIBRARY_COMMAND_ARRAY_SIZE; // first vendor command
            max = COMMAND_COUNT - 1; // last vendor command
            diff = 1; // needs initialization to keep
            // compiler happy

            while(min <= max)
            {
```
commandIndex = (min + max + 1) / 2;
diff = GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex)
    - searchIndex;
if(diff == 0)
    return commandIndex;
if(diff > 0)
    max = commandIndex - 1;
else
    min = commandIndex + 1;
}
// didn't find and exact match. commandIndex will be pointing at the last
// item tested. If 'diff' is positive, then the last item tested was
// larger index of the command code so it is the smallest value
// larger than the requested value.
if(diff > 0)
    return commandIndex;
// if 'diff' is negative, then the value tested was smaller than
// the commandCode index and the next higher value is the correct one.
// Note: this will necessarily be in range because of the earlier check
// that the index was within range.
return commandIndex + 1;
#else
    // If there are no vendor commands so anything with the vendor bit set is out
    // of range
    return UNIMPLEMENTED_COMMAND_INDEX;
#endif
// Get here if the V-Bit was not set in 'commandCode'
if(GET_ATTRIBUTE(s_ccAttr[LIBRARY_COMMAND_ARRAY_SIZE - 1], TPMA_CC,
    commandIndex) < searchIndex)
{
    // requested index is out of the range to the top
    #if VENDOR_COMMAND_ARRAY_SIZE > 0
        // If there are vendor commands, then the first vendor command
        // is the next value greater than the commandCode.
        // NOTE: we got here if the starting index did not have the V bit but we
        // reached the end of the array of library commands (non-vendor). Since
        // there is at least one vendor command, and vendor commands are always
        // in a compressed list that starts after the library list, the next
        // index value contains a valid vendor command.
        return LIBRARY_COMMAND_ARRAY_SIZE;
    #else
        // if there are no vendor commands, then this is out of range
        return UNIMPLEMENTED_COMMAND_INDEX;
    #endif
    // If the request is lower than any value in the array, then return
    // the lowest value (needs to be an index for an implemented command
    if(GET_ATTRIBUTE(s_ccAttr[0], TPMA_CC, commandIndex) >= searchIndex)
    {
        return NextImplementedIndex(0);
    }
else
{
    #if COMPRESSION_LISTS
        COMMAND_INDEX commandIndex = UNIMPLEMENTED_COMMAND_INDEX;
        COMMAND_INDEX min = 0;
        COMMAND_INDEX max = LIBRARY_COMMAND_ARRAY_SIZE - 1;
        int diff = 1;
        #if LIBRARY_COMMAND_ARRAY_SIZE == 0
            #error "Something is terribly wrong"
        #endif
        // The s_ccAttr array contains an extra entry at the end (a zero value).
        // Don't count this as an array entry. This means that max should start
        // out pointing to the last valid entry in the array which is - 2
        //}
pAssert(max == (sizeof(s_ccAttr) / sizeof(TPMA_CC)
   - VENDOR_COMMAND_ARRAY_SIZE - 2));
while(min <= max)
{
    commandIndex = (min + max + 1) / 2;
    diff = GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex) - searchIndex;
    if(diff == 0)
    {
        return commandIndex;
    }
    if(diff > 0)
    {
        max = commandIndex - 1;
    }
    else
    {
        min = commandIndex + 1;
    }
}
// didn't find and exact match. commandIndex will be pointing at the
// last item tested. If diff is positive, then the last item tested was
// larger index of the command code so it is the smallest value
// larger than the requested value.
if(diff > 0)
    return commandIndex;
// if diff is negative, then the value tested was smaller than
// the commandCODE index and the next higher value is the correct one.
// Note: this will necessarily be in range because of the earlier check
// that the index was within range.
return commandIndex + 1;
#else
    // The list is not compressed so offset into the array by the command
    // code value of the first entry in the list. Then go find the first
    // implemented command.
    return NextImplementedIndex(searchIndex
    - (COMMAND_INDEX)s_ccAttr[0].commandIndex);
#endif
}

9.3.3.3 CommandCodeToComandIndex()

This function returns the index in the various attributes arrays of the command.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIMPLEMENTED_COMMAND_INDEX</td>
<td>command is not implemented</td>
</tr>
<tr>
<td>other</td>
<td>index of the command</td>
</tr>
</tbody>
</table>

COMMAND_INDEX
CommandCodeToComandIndex(TPM_CC commandCode // IN: the command code to look up)
{
    // Extract the low 16-bits of the command code to get the starting search index
    COMMAND_INDEX searchIndex = (COMMAND_INDEX)commandCode;
    BOOL vendor = (commandCode & CC_VEND) != 0;
    COMMAND_INDEX commandIndex;
    #if !COMPRESSED_LISTS
    if(!vendor)
    {
        commandIndex = searchIndex - (COMMAND_INDEX)s_ccAttr[0].commandIndex;
        // Check for out of range or unimplemented.
        // Note, since a COMMAND_INDEX is unsigned, if searchIndex is smaller than
        // the lowest value of command, it will become a 'negative' number making
        // it look like a large unsigned number, this will cause it to fail
        // the unsigned check below.
        if(commandIndex >= LIBRARY_COMMAND_ARRAY_SIZE
        return commandIndex;
    }
    #endif
}
198  || (s_commandAttributes[commandIndex] & IS_IMPLEMENTED) == 0)
199  return UNIMPLEMENTED_COMMAND_INDEX;
200  return commandIndex;
201  }
202 #endif
203 // Need this code for any vendor code lookup or for compressed lists
204 commandIndex = GetClosestCommandIndex(commandCode);
205
206 // Look at the returned value from get closest. If it isn't the one that was
207 // requested, then the command is not implemented.
208 if(commandIndex != UNIMPLEMENTED_COMMAND_INDEX)
209 {
210    if((GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex)
211        != searchIndex)
212        || (IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V)) != vendor)
213        commandIndex = UNIMPLEMENTED_COMMAND_INDEX;
214    }
215  return commandIndex;
216 }

9.3.3.4 GetNextCommandIndex()

This function returns the index of the next implemented command.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNIMPLEMENTED_COMMAND_INDEX</td>
<td>no more implemented commands</td>
</tr>
<tr>
<td>other</td>
<td>the index of the next implemented command</td>
</tr>
</tbody>
</table>

217 COMMAND_INDEX
218 GetNextCommandIndex(
219    COMMAND_INDEX    commandIndex    // IN: the starting index
220 )
221 {
222    while(++commandIndex < COMMAND_COUNT)
223    {
224        #if !COMPRESSED_LISTS
225        if(s_commandAttributes[commandIndex] & IS_IMPLEMENTED)
226        #endif
227        return commandIndex;
228    }
229  return UNIMPLEMENTED_COMMAND_INDEX;
230 }

9.3.3.5 GetCommandCode()

This function returns the commandCode associated with the command index

231 TPM_CC
232 GetCommandCode(
233    COMMAND_INDEX    commandIndex    // IN: the command index
234 )
235 {
236    TPM_CC    commandCode = GET_ATTRIBUTE(s_ccAttr[commandIndex],
237                                              TPMA_CC, commandIndex);
238    if(IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V))
239        commandCode += CC_VEND;
240  return commandCode;
241 }
9.3.3.6 CommandAuthRole()

This function returns the authorization role required of a handle.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTH_NONE</td>
<td>no authorization is required</td>
</tr>
<tr>
<td>AUTH_USER</td>
<td>user role authorization is required</td>
</tr>
<tr>
<td>AUTH_ADMIN</td>
<td>admin role authorization is required</td>
</tr>
<tr>
<td>AUTH_DUP</td>
<td>duplication role authorization is required</td>
</tr>
</tbody>
</table>

AUTH_ROLE

CommandAuthRole(
COMND_INDEX commandIndex, // IN: command index
UINT32 handleIndex // IN: handle index (zero based)
)
{
    if(0 == handleIndex)
    {
        // Any authorization role set?
        COMND_ATTRIBUTES properties = s_commandAttributes[commandIndex];
        if(properties & HANDLE_1_USER)
            return AUTH_USER;
        if(properties & HANDLE_1_ADMIN)
            return AUTH_ADMIN;
        if(properties & HANDLE_1_DUP)
            return AUTH_DUP;
    }
    else if(1 == handleIndex)
    {
        if(s_commandAttributes[commandIndex] & HANDLE_2_USER)
            return AUTH_USER;
    }
    return AUTH_NONE;
}

9.3.3.7 EncryptSize()

This function returns the size of the decrypt size field. This function returns 0 if encryption is not allowed.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>encryption not allowed</td>
</tr>
<tr>
<td>2</td>
<td>size field is two bytes</td>
</tr>
<tr>
<td>4</td>
<td>size field is four bytes</td>
</tr>
</tbody>
</table>

int

EncryptSize(
COMND_INDEX commandIndex // IN: command index
)
{
    return ((s_commandAttributes[commandIndex] & ENCRYPT_2) ? 2 :
            (s_commandAttributes[commandIndex] & ENCRYPT_4) ? 4 : 0);
}
9.3.3.8 DecryptSize()

This function returns the size of the decrypt size field. This function returns 0 if decryption is not allowed.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>encryption not allowed</td>
</tr>
<tr>
<td>2</td>
<td>size field is two bytes</td>
</tr>
<tr>
<td>4</td>
<td>size field is four bytes</td>
</tr>
</tbody>
</table>

```c
int DecryptSize(
    COMMAND_INDEX commandIndex   // IN: command index
)
{
    return ((s_commandAttributes[commandIndex] & DECRYPT_2) ? 2 :
        (s_commandAttributes[commandIndex] & DECRYPT_4) ? 4 : 0);
}
```

9.3.3.9 IsSessionAllowed()

This function indicates if the command is allowed to have sessions.
This function must not be called if the command is not known to be implemented.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>session is allowed with this command</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>session is not allowed with this command</td>
</tr>
</tbody>
</table>

```c
BOOL IsSessionAllowed(
    COMMAND_INDEX commandIndex   // IN: the command to be checked
)
{
    return ((s_commandAttributes[commandIndex] & NO_SESSIONS) == 0);
}
```

9.3.3.10 IsHandleInResponse()

This function determines if a command has a handle in the response.

```c
BOOL IsHandleInResponse(
    COMMAND_INDEX commandIndex   // IN: Command to check
)
{
    return ((s_commandAttributes[commandIndex] & R_HANDLE) != 0);
}
```

9.3.3.11 IsWriteOperation()

Checks to see if an operation will write to an NV Index and is subject to being blocked by read-lock.

```c
BOOL IsWriteOperation(
    COMMAND_INDEX commandIndex   // IN: Command to check
)
```
301 {  
302 #ifdef WRITE_LOCK  
303 return ((s_commandAttributes[commandIndex] & WRITE_LOCK) != 0);  
304 #else  
305 if(!IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V))  
306 {  
307     switch(GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex))  
308     {  
309         case TPM_CC_NV_Write:  
310             #if CC_NV_Increment  
311                 case TPM_CC_NV_Increment:  
312                 #endif  
313                 #if CC_NV_SetBits  
314                     case TPM_CC_NV_SetBits:  
315                 #endif  
316                 #if CC_NV_Extend  
317                     case TPM_CC_NV_Extend:  
318                 #endif  
319                 #if CC_AC_Send  
320                     case TPM_CC_AC_Send:  
321                 #endif  
322                     // NV write lock counts as a write operation for authorization purposes.  
323                     // We check to see if the NV is write locked before we do the  
324                     // authorization. If it is locked, we fail the command early.  
325                     case TPM_CC_NV_WriteLock:  
326                         return TRUE;  
327                     default:  
328                         break;  
329                 }  
330 }  
331 return FALSE;  
332 #endif  
333 }

9.3.3.12 IsReadOperation()  

Checks to see if an operation will write to an NV Index and is subject to being blocked by write-lock.

334 BOOL
335 IsReadOperation(  
336     COMMAND_INDEX commandIndex  // IN: Command to check  
337 )  
338 {  
339 #ifdef READ_LOCK  
340 return ((s_commandAttributes[commandIndex] & READ_LOCK) != 0);  
341 #else  
342 if(!IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V))  
343 {  
344     switch(GET_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, commandIndex))  
345     {  
346         case TPM_CC_NV_Read:  
347         case TPM_CC_PolicyNV:  
348         case TPM_CC_NV_Certify:  
349             // NV read lock counts as a read operation for authorization purposes.  
350             // We check to see if the NV is read locked before we do the  
351             // authorization. If it is locked, we fail the command early.  
352             case TPM_CC_NV_ReadLock:  
353                 return TRUE;  
354             default:  
355                 break;  
356         }  
357     }  
358 return FALSE;  
359 }
9.3.3.13 CommandCapGetCCList()

This function returns a list of implemented commands and command attributes starting from the command in commandCode.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>more command attributes are available</td>
</tr>
<tr>
<td>NO</td>
<td>no more command attributes are available</td>
</tr>
</tbody>
</table>

```c
TPMI_YES_NO CommandCapGetCCList(
    TPM_CC commandCode, // IN: start command code
    UINT32 count,      // IN: maximum count for number of entries in 'commandList'
    TPML_CCA *commandList // OUT: list of TPMA_CC
)
{
    TPMI_YES_NO more = NO;
    COMMAND_INDEX commandIndex;

    // initialize output handle list count
    commandList->count = 0;
    for(commandIndex = GetClosestCommandIndex(commandCode);
        commandIndex != UNIMPLEMENTED_COMMAND_INDEX;
        commandIndex = GetNextCommandIndex(commandIndex))
    {
        #if !COMPRESSED_LISTS
            // this check isn't needed for compressed lists.
            if(!s_commandAttributes[commandIndex] & IS_IMPLEMENTED))
                continue;
        #endif

        if(commandList->count < count)
            {
                // If the list is not full, add the attributes for this command.
                commandList->commandAttributes[commandList->count] = s_cc Attr[commandIndex];
                commandList->count++;
            }
        else
            {
                // If the list is full but there are more commands to report,
                // indicate this and return.
                more = YES;
                break;
            }
    }
    return more;
}
```

9.3.3.14 IsVendorCommand()

Function indicates if a command index references a vendor command.
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>command is a vendor command</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>command is not a vendor command</td>
</tr>
</tbody>
</table>

```c
402    BOOL  IsVendorCommand(
403    COMMAND_INDEX  commandIndex    // IN: command index to check
405    )
406    {
407    return (IS_ATTRIBUTE(s_ccAttr[commandIndex], TPMA_CC, V));
408    }
```
9.4 Entity.c

9.4.1 Description

The functions in this file are used for accessing properties for handles of various types. Functions in other files require handles of a specific type but the functions in this file allow use of any handle type.

9.4.2 Includes

```c
#include "Tpm.h"
```

9.4.3 Functions

9.4.3.1 EntityGetLoadStatus()

This function will check that all the handles access loaded entities.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HANDLE</td>
<td>handle type does not match</td>
</tr>
<tr>
<td>TPM_RCREFERENCE_Hx()</td>
<td>entity is not present</td>
</tr>
<tr>
<td>TPM_RC_HIERARCHY</td>
<td>entity belongs to a disabled hierarchy</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>handle is an evict object but there is no space to load it to RAM</td>
</tr>
</tbody>
</table>

```c
TPM_RC
EntityGetLoadStatus(  
  COMMAND *command   // IN/OUT: command parsing structure  
)  
{
  UINT32 i;
  TPM_RC result = TPM_RC_SUCCESS;
  for(i = 0; i < command->handleNum; i++)
  {  
    TPM_HANDLE handle = command->handles[i];
    switch(HandleGetType(handle))
    {
      // For handles associated with hierarchies, the entity is present
      // only if the associated enable is SET.
      case TPM_HT_PERMANENT:
        switch(handle)
        {
          case TPM_RH_OWNER:
            if(!gc.shEnable)
              result = TPM_RC_HIERARCHY;
            break;
          #ifdef VENDOR_PERMANENT
          case VENDOR_PERMANENT:
            #endif
          case TPM_RH_ENDORSEMENT:
            if(!gc.ehEnable)
              result = TPM_RC_HIERARCHY;
            break;
          case TPM_RH_PLATFORM:
            if(!g_phEnable)
              result = TPM_RC_HIERARCHY;
            break;
        }
    }
  }
```

29  #ifdef VENDOR_PERMANENT
30  case VENDOR_PERMANENT:
31  #endif
32  case TPM_RH_ENDORSEMENT:
33  if(!gc.ehEnable)
34  result = TPM_RC_HIERARCHY;
35  break;
36  case TPM_RH_PLATFORM:
37  if(!g_phEnable)
38  result = TPM_RC_HIERARCHY;
39  break;
40  ```
36 // null handle, PW session handle and lockout
37 // handle are always available
38 case TPM_RH_NULL:
39 case TPM_RS_PW:
40 // Need to be careful for lockout. Lockout is always available
41 // for policy checks but not always available when authValue
42 // is being checked.
43 case TPM_RH_LOCKOUT:
44 break;
45 default:
46 // handling of the manufacture specific handles
47 if(((TPM_RH)handle >= TPM_RH_AUTH_00)
48 && ((TPM_RH)handle <= TPM_RH_AUTH_FF))
49 // use the value that would have been returned from
50 // unmarshaling if it did the handle filtering
51 result = TPM_RC_VALUE;
52 else
53 FAIL(FATAL_ERROR_INTERNAL);
54 break;
55 }
56 break;
57 case TPM_HT_TRANSIENT:
58 // For a transient object, check if the handle is associated
59 // with a loaded object.
60 if(!IsObjectPresent(handle))
61 result = TPM_RC_REFERENCE_H0;
62 break;
63 case TPM_HT_PERSISTENT:
64 // Persistent object
65 // Copy the persistent object to RAM and replace the handle with the
66 // handle of the assigned slot. A TPM_RC_OBJECT_MEMORY,
67 // TPM_RC_HIERARCHY or TPM_RC_REFERENCE_H0 error may be returned by
68 // ObjectLoadEvict()
69 result = ObjectLoadEvict(&command->handles[i], command->index);
70 break;
71 case TPM_HT_HMAC_SESSION:
72 // For an HMAC session, see if the session is loaded
73 // and if the session in the session slot is actually
74 // an HMAC session.
75 if(SessionIsLoaded(handle))
76 {
77 SESSION *session;
78 session = SessionGet(handle);
79 // Check if the session is a HMAC session
80 if(session->attributes.isPolicy == SET)
81 result = TPM_RC_HANDLE;
82 }
83 else
84 result = TPM_RC_REFERENCE_H0;
85 break;
86 case TPM_HT_POLICY_SESSION:
87 // For a policy session, see if the session is loaded
88 // and if the session in the session slot is actually
89 // a policy session.
90 if(SessionIsLoaded(handle))
91 {
92 SESSION *session;
93 session = SessionGet(handle);
94 // Check if the session is a policy session
95 if(session->attributes.isPolicy == CLEAR)
96 result = TPM_RC_HANDLE;
97 }
98 else
99 result = TPM_RC_REFERENCE_H0;
100 break;
101 case TPM_HT_NV_INDEX:
// For an NV Index, use the TPM-specific routine
// to search the IN Index space.
result = NvIndexIsAccessible(handle);
break;
case TPM_HT_PCR:
// Any PCR handle that is unmarshaled successfully referenced
// a PCR that is defined.
break;
#if CC_AC_Send
 case TPM_HT_AC:
// Use the TPM-specific routine to search for the AC
result = AcIsAccessible(handle);
break;
#endif
default:
// Any other handle type is a defect in the unmarshaling code.
FAIL(FATAL_ERROR_INTERNAL);
break;
}
if(result != TPM_RC_SUCCESS)
{
if(result == TPM_RC_REFERENCE_H0)
    result = result + i;
else
    result = RcSafeAddToResult(result, TPM_RC_H + g_rcIndex[i]);
break;
}
return result;
}

9.4.3.2 EntityGetAuthValue()

This function is used to access the authValue associated with a handle. This function assumes that the handle references an entity that is accessible and the handle is not for a persistent objects. That is EntityGetLoadStatus() should have been called. Also, the accessibility of the authValue should have been verified by IsAuthValueAvailable().

This function copies the authorization value of the entity to auth.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>number of bytes in the authValue with 0's stripped</td>
</tr>
</tbody>
</table>

UINT16 EntityGetAuthValue(
                      TPM_ DH_ENTITY   handle,       // IN: handle of entity
                      TPM2B_AUTH      *auth        // OUT: authValue of the entity
                  )
{
    TPM2B_AUTH      *pAuth = NULL;
    auth->t.size = 0;
    switch(HandleGetType(handle))
    {
        case TPM_HT_PERMANENT:
        {
            switch(handle)
            {
            case TPM_RH_OWNER:
                // ownerAuth for TPM_RH_OWNER
                pAuth = &g.p.ownerAuth;
                break;
            }
case TPM_RH_ENDORSEMENT:
   // endorsementAuth for TPM_RH_ENDORSEMENT
   pAuth = &gp.endorsementAuth;
   break;

case TPM_RH_PLATFORM:
   // platformAuth for TPM_RH_PLATFORM
   pAuth = &gc.platformAuth;
   break;

case TPM_RH_LOCKOUT:
   // lockoutAuth for TPM_RH_LOCKOUT
   pAuth = &gp.lockoutAuth;
   break;

case TPM_RH_NULL:
   // nullAuth for TPM_RH_NULL. Return 0 directly here
   return 0;
   break;

#ifdef VENDOR_PERMANENT
    case VENDOR_PERMANENT:
        // vendor authorization value
        pAuth = &g_platformUniqueDetails;
#else
    default:
        // If any other permanent handle is present it is
        // a code defect.
        FAIL(FATAL_ERROR_INTERNAL);
        break;
#endif
}
break;

case TPM_HT_TRANSIENT:
   // authValue for an object
   // A persistent object would have been copied into RAM
   // and would have an transient object handle here.
   
   OBJECT *object;
   
   object = HandleToObject(handle);
   // special handling if this is a sequence object
   if(ObjectIsSequence(object))
      {
         pAuth = &((HASH_OBJECT *)object)->auth;
      }
   else
      {
         // Authorization is available only when the private portion of
         // the object is loaded. The check should be made before
         // this function is called
         pAssert(object->attributes.publicOnly == CLEAR);
         pAuth = &object->sensitive.authValue;
      }
   break;

case TPM_HT_NV_INDEX:
   // authValue for an NV index
   
   NV_INDEX *nvIndex = NvGetIndexInfo(handle, NULL);
   pAssert(nvIndex != NULL);
   pAuth = &nvIndex->authValue;
   break;

case TPM_HT_PCR:
   // authValue for PCR
   pAuth = PCRGetAuthValue(handle);
   break;

default:
   // If any other handle type is present here, then there is a defect
218         // in the unmarshaling code.
219         FAIL(FATAL_ERROR_INTERNAL);
220         break;
221     }
222     // Copy the authValue
223     MemoryCopy2B(&auth->b, &pAuth->b, sizeof(auth->t.buffer));
224     MemoryRemoveTrailingZeros(auth);
225     return auth->t.size;
226 }

9.4.3.3 EntityGetAuthPolicy()

This function is used to access the authPolicy associated with a handle. This function assumes that the handle references an entity that is accessible and the handle is not for a persistent objects. That is EntityGetLoadStatus() should have been called. Also, the accessibility of the authPolicy should have been verified by IsAuthPolicyAvailable().

This function copies the authorization policy of the entity to authPolicy.

The return value is the hash algorithm for the policy.

227 #define TPM_ALG_HASH
228 extern TPM2B_DIGEST EntityGetAuthPolicy(
229     TPMI_DH_ENTITY   handle,    // IN: handle of entity
230     TPM2B_DIGEST    *authPolicy  // OUT: authPolicy of the entity
231 )
232 {
233     TPMI_ALG_HASH       hashAlg = TPM_ALG_NULL;
234     authPolicy->t.size = 0;
235     switch(HandleGetType(handle))
236     {
237         case TPM_HT_PERMANENT:
238             switch(handle)
239             {
240                 case TPM_RH_OWNER:
241                     // ownerPolicy for TPM RH OWNER
242                     *authPolicy = gp.ownerPolicy;
243                     hashAlg = gp.ownerAlg;
244                     break;
245                 case TPM_RH_ENDORSEMENT:
246                     // endorsementPolicy for TPM RH ENDORSEMENT
247                     *authPolicy = gp.endorsementPolicy;
248                     hashAlg = gp.endorsementAlg;
249                     break;
250                 case TPM_RH_PLATFORM:
251                     // platformPolicy for TPM RH PLATFORM
252                     *authPolicy = gc.platformPolicy;
253                     hashAlg = gc.platformAlg;
254                     break;
255                 case TPM_RH_LOCKOUT:
256                     // lockoutPolicy for TPM RH LOCKOUT
257                     *authPolicy = gp.lockoutPolicy;
258                     hashAlg = gp.lockoutAlg;
259                     break;
260                 default:
261                     return TPM_ALG_ERROR;
262                     break;
263             }
264         break;
265     case TPM_HT_TRANSIENT:
266         // authPolicy for an object
267         {
268             OBJECT *object = HandleToObject(handle);
*authPolicy = object->publicArea.authPolicy;
hashAlg = object->publicArea.nameAlg;
}
break;
case TPM_HT_NV_INDEX:
  // authPolicy for a NV index
  {
    NV_INDEX *nvIndex = NvGetIndexInfo(handle, NULL);
pAssert(nvIndex != 0);
    *authPolicy = nvIndex->publicArea.authPolicy;
hashAlg = nvIndex->publicArea.nameAlg;
  }
break;
case TPM_HT_PCR:
  // authPolicy for a PCR
  hashAlg = PCRGetAuthPolicy(handle, authPolicy);
b break;
default:
  // If any other handle type is present it is a code defect.
  FAIL(FATAL_ERROR_INTERNAL);
b break;
  }
return hashAlg;
}

9.4.3.4 EntityGetName()

This function returns the Name associated with a handle.

TPM2B_NAME *
EntityGetName(
  TPMI_DH_ENTITY handle, // IN: handle of entity
  TPM2B_NAME      *name  // OUT: name of entity
)
{
  switch(HandleGetType(handle))
  {
    case TPM_HT_TRANSIENT:
    {
      // Name for an object
      OBJECT *object = HandleToObject(handle);
      // an object with no nameAlg has no name
      if(object->publicArea.nameAlg == TPM_ALG_NULL)
        name->b.size = 0;
      else
        *name = object->name;
b break;
    }
    case TPM_HT_NV_INDEX:
    // Name for a NV index
    NvGetNameByIndexHandle(handle, name);
b break;
    default:
    // For all other types, the handle is the Name
    name->t.size = sizeof(TPM_HANDLE);
    UINT32_TO_BYTE_ARRAY(handle, name->t.name);
b break;
      }
  return name;
}
9.4.3.5 EntityGetHierarchy()

This function returns the hierarchy handle associated with an entity.

a) A handle that is a hierarchy handle is associated with itself.

b) An NV index belongs to TPM_RHPLATFORM if TPMA_NVPLATFORMCREATE is SET, otherwise it belongs to TPM_RH_OWNER.

c) An object handle belongs to its hierarchy.

```c
TPMI_RH_HIERARCHY
EntityGetHierarchy(
    TPMI_DH_ENTITY   handle   // IN :handle of entity
) )
{ 
    TPMI_RH_HIERARCHY hierarchy = TPM_RH_NULL;
    switch(HandleGetType(handle))
    {
        case TPM_HT_PERMANENT:
            // hierarchy for a permanent handle
            switch(handle)
            {
                case TPM_RH_PLATFORM:
                case TPM_RH_ENDORSEMENT:
                case TPM_RH_NULL:
                    hierarchy = handle;
                    break;
                // all other permanent handles are associated with the owner
                // hierarchy. (should only be TPM_RH_OWNER and TPM_RH_LOCKOUT)
                default:
                    hierarchy = TPM_RH_OWNER;
                    break;
                }
            break;
        case TPM_HT_NV_INDEX:
            // hierarchy for NV index
            
            // If only the platform can delete the index, then it is
            // considered to be in the platform hierarchy, otherwise it
            // is in the owner hierarchy.
            if(IS_ATTRIBUTE(nvIndex->publicArea.attributes, TPMA_NV, PLATFORMCREATE))
                hierarchy = TPM_RH_PLATFORM;
            else
                hierarchy = TPM_RH_OWNER;
            break;
        case TPM_HT_TRANSIENT:
            // hierarchy for an object
            
            // If object has PPS hierarchy, then it is
            if(object->attributes.ppsHierarchy)
            { hierarchy = TPM_RH_PLATFORM;
            }
            else if(object->attributes.epsHierarchy)
            { hierarchy = TPM_RH_ENDORSEMENT;
            }
            else if(object->attributes.spsHierarchy)
            { hierarchy = TPM_RH_OWNER;
            }
        break;
    }
```
{ 
    hierarchy = TPM_RH_OWNER;
}

break;
case TPM_HT_PCR:
    hierarchy = TPM_RH_OWNER;
    break;
default:
    FAIL(FATAL_ERROR_INTERNAL);
    break;
}

// this is unreachable but it provides a return value for the default case which makes the compiler happy
return hierarchy;
9.5 Global.c

9.5.1 Description

This file will instance the TPM variables that are not stack allocated. The descriptions for these variables are in Global.h.

9.5.2 Includes and Defines

```c
#define GLOBAL_C
#include "Tpm.h"
```

9.5.3 __IGNORE_STATE__

This macro value is used to delimit values that are only used within the scope of a single TPM command. For a virtualized TPM, this would be state that does not need to be preserved between commands. This is not a lot of data and it may not be necessary to exclude this data.

9.5.4 Global Data Values

These values are visible across multiple modules.

```c
BOOL g_phEnable;
const UINT16 g_rcIndex[15] = {TPM_RC_1, TPM_RC_2, TPM_RC_3, TPM_RC_4,
                              TPM_RC_5, TPM_RC_6, TPM_RC_7, TPM_RC_8,
                              TPM_RC_9, TPM_RC_A, TPM_RC_B, TPM_RC_C,
                              TPM_RC_D, TPM_RC_E, TPM_RC_F};
TPM_HANDLE g_exclusiveAuditSession;
UINT64 g_time;
#if CLOCK_STOPS
CLOCK_NONCE g_timeEpoch;
#endif
BOOL g_pcrReConfig;
TPMI_DH_OBJECT g_DRTMHandle;
BOOL g_DrtmPreStartup;
BOOL g_StartupLocality3;
#if USE_DA_USED
BOOL g_daUsed;
#endif
BOOL g_powerWasLost;
BOOL g_clearOrderly;
TPM_SU g_prevOrderlyState;
UPDATE_TYPE g_updateNV;
BOOL g_nvOk;
TPM_RC g_NvStatus;
TPM2B_AUTH g_platformUniqueDetails;
ALGORITHM_VECTOR g_implementedAlgorithms;
ALGORITHM_VECTOR g_toTest;
CRYPTO_SELF_TEST_STATE g_cryptoSelfTestState;  // This structure contains the
                                                  // cryptographic self-test
#if SIMULATION
BOOL g_forceFailureMode;
#endif
BOOL g_inFailureMode;
STATE_CLEAR_DATA gc;
STATE_RESET_DATA gr;
PERSISTENT_DATA gp;
ORDERLY_DATA go;
```
9.5.5 Private Values

9.5.5.1 Used in SessionProcess.c

```c
#ifndef __IGNORE_STATE__ // DO NOT DEFINE THIS VALUE

 These values do not need to be retained between commands.

 TPM_HANDLE s_sessionHandles[MAX_SESSION_NUM];
 TPM_ATTRIBUTES s_attributes[MAX_SESSION_NUM];
 TPM_ATTRIBUTES s_associatedHandles[MAX_SESSION_NUM];
 TPM2B_NONCE s_nonceCaller[MAX_SESSION_NUM];
 TPM2B_AUTH s_inputAuthValues[MAX_SESSION_NUM];
 SESSION *s_usedSessions[MAX_SESSION_NUM];
 UINT32 s_encryptSessionIndex;
 UINT32 s_decryptSessionIndex;
 UINT32 s_auditSessionIndex;

 #endif // __IGNORE_STATE__

 BOOL s_DAPendingOnNV;

 #if CC_GetCommandAuditDigest

 TPM2B_DIGEST s_cpHashForCommandAudit;

 #endif

9.5.5.2 Used in DA.c

```c
#ifndef __IGNORE_STATE__ // DO NOT DEFINE THIS VALUE

 UINT64 s_selfHealTimer;
 UINT64 s_lockoutTimer;

 #endif // !ACCUMULATE_SELF_HEAL_TIMER

9.5.5.3 Used in NV.c

```c
UINT64 s_maxCounter;
NV_REF s_evictNvEnd;
BYTE s_indexOrderlyRam[RAM_INDEX_SPACE];

 #ifndef __IGNORE_STATE__ // DO NOT DEFINE THIS VALUE

 NV_INDEX s_cachedNvIndex;
 NV_REF s_cachedNvRef;
 BYTE *s_cachedNvRamRef;

 #endif // __IGNORE_STATE__

9.5.5.4 Used in Object.c

```c
OBJECT s_objects[MAX_LOADED_OBJECTS];

9.5.5.5 Used in PCR.c

```c
PCR s_pcrs[IMPLEMENTATION_PCR];

9.5.5.6 Used in Session.c

```c
SESSION_SLOT s_sessions[MAX_LOADED_SESSIONS];
 UINT32 s_oldestSavedSession;
 int s_freeSessionSlots;

9.5.5.7 Used in MemoryLib.c

```c
#ifndef __IGNORE_STATE__ // DO NOT DEFINE THIS VALUE

```
9.5.5.8 used in TpmFail.c

UINT32 s_failFunction;
UINT32 s_failLine;
UINT32 s_failCode;

9.5.5.9 Used in CryptRand.c

This is the state used when the library uses a random number generator. A special function is installed for the library to call. That function picks up the state from this location and uses it for the generation of the random number.

RAND_STATE *s_random;

9.5.5.10 Used in Manufacture.c

The values is here rather than in the simulator or platform files in order to make it easier to find the TPM state. This is significant when trying to do TPM virtualization when the TPM state has to be moved along with virtual machine with which it is associated.

BOOL g_manufactured = FALSE;

9.5.5.11 Used in Power.c

This is here for the same reason that g_manufactured is here. Both of these values can be provided by the actual platform-specific code or by hardware indications.

BOOL g_initialized;

9.5.5.12 Purpose-specific String Constants

These string constants are shared across functions to make sure that they are all using consistent sting values.

TPM2B_STRING(PRIMARY_OBJECT_CREATION, "Primary Object Creation");
TPM2B_STRING(CFB_KEY, "CFB");
TPM2B_STRING(CONTEXT_KEY, "CONTEXT");
TPM2B_STRING(INTEGRITY_KEY, "INTEGRITY");
TPM2B_STRING(SECRET_KEY, "SECRET");
TPM2B_STRING(SESSION_KEY, "ATH");
TPM2B_STRING(STORAGE_KEY, "STORAGE");
TPM2B_STRING(XOR_KEY, "XOR");
TPM2B_STRING(COMMIT_STRING, "ECDAA Commit");
TPM2B_STRING(DUPLICATE_STRING, "DUPLICATE");
TPM2B_STRING(IDENTITY_STRING, "IDENTITY");
TPM2B_STRING(OBFUSCATE_STRING, "OBFUSCATE");
# if SELF_TEST
TPM2B_STRING(OAEP_TEST_STRING, "OAEP Test Value");
# endif // SELF_TEST
9.6 Handle.c

9.6.1 Description
This file contains the functions that return the type of a handle.

9.6.2 Includes
1 #include "Tpm.h"

9.6.3 Functions

9.6.3.1 HandleGetType()
This function returns the type of a handle which is the MSO of the handle.

2 TPM_HT
3 HandleGetType(
4     TPM_HANDLE handle // IN: a handle to be checked
5 )
6 {
7     // return the upper bytes of input data
8     return (TPM_HT)((handle & HR_RANGE_MASK) >> HR_SHIFT);
9 }

9.6.3.2 NextPermanentHandle()
This function returns the permanent handle that is equal to the input value or is the next higher value. If
there is no handle with the input value and there is no next higher value, it returns 0:

10 TPM_HANDLE
11 NextPermanentHandle(
12     TPM_HANDLE inHandle // IN: the handle to check
13 )
14 {
15     // If inHandle is below the start of the range of permanent handles
16     // set it to the start and scan from there
17     if(inHandle < TPM_RH_FIRST)
18         inHandle = TPM_RH_FIRST;
19     // scan from input value until we find an implemented permanent handle
20     // or go out of range
21     for(; inHandle <= TPM_RH_LAST; inHandle++)
22     {
23         switch(inHandle)
24         {
25             case TPM_RH_OWNER:
26             case TPM_RH_NULL:
27             case TPM_RS_PW:
28             case TPM_RH_LOCKOUT:
29             case TPM_RH_ENDORSEMENT:
30             case TPM_RH_PLATFORM:
31             case TPM_RH_PLATFORM_NV:
32             #ifdef VENDOR_PERMANENT
33                 case VENDOR_PERMANENT:
34             #endif
35             return inHandle;
36             break;
37             default:
38                 break;
9.6.3.3 PermanentCapGetHandles()

This function returns a list of the permanent handles of PCR, started from handle. If handle is larger than the largest permanent handle, an empty list will be returned with more set to NO.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more handles available</td>
</tr>
<tr>
<td>NO</td>
<td>all the available handles has been returned</td>
</tr>
</tbody>
</table>

```
TPMI_YES_NO PermanentCapGetHandles(
TPM_HANDLE handle,       // IN: start handle
UINT32 count,            // IN: count of returned handles
TPML_HANDLE *handleList  // OUT: list of handle
) {
  TPMI_YES_NO more = NO;
  UINT32 i;
  pAssert(HandleGetType(handle) == TPM_HT_PERMANENT);
  // Initialize output handle list
  handleList->count = 0;
  // The maximum count of handles we may return is MAX_CAP_HANDLES
  if(count > MAX_CAP_HANDLES) count = MAX_CAP_HANDLES;
  // Iterate permanent handle range
  for(i = NextPermanentHandle(handle);
      i != 0; i = NextPermanentHandle(i + 1))
  {
    if(handleList->count < count)
    {
      // If we have not filled up the return list, add this permanent handle to it
      handleList->handle[handleList->count] = i;
      handleList->count++;
    }
  }
  else
  {
    // If the return list is full but we still have permanent handle available, report this and stop iterating
    more = YES;
    break;
  }
  return more;
}
```

9.6.3.4 PermanentHandleGetPolicy()

This function returns a list of the permanent handles of PCR, started from handle. If handle is larger than the largest permanent handle, an empty list will be returned with more set to NO.
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if there are more handles available</td>
</tr>
<tr>
<td>NO</td>
<td>all the available handles has been returned</td>
</tr>
</tbody>
</table>

```c
TPMI_YES_NO
PermanentHandleGetPolicy(
    TPM_HANDLE handle, // IN: start handle
    UINT32 count,     // IN: max count of returned handles
    TPML_TAGGED_POLICY *policyList // OUT: list of handle
)
{
    TPMI_YES_NO more = NO;
    pAssert(HandleGetType(handle) == TPM_HT_PERMANENT);
    // Initialize output handle list
    policyList->count = 0;
    // The maximum count of policies we may return is MAX_TAGGED_POLICIES
    if(count > MAX_TAGGED_POLICIES)
        count = MAX_TAGGED_POLICIES;
    // Iterate permanent handle range
    for(handle = NextPermanentHandle(handle);
        handle != 0;
        handle = NextPermanentHandle(handle + 1))
    {
        TPM2B_DIGEST policyDigest;
        TPM_ALG_ID policyAlg;
        // Check to see if this permanent handle has a policy
        policyAlg = EntityGetAuthPolicy(handle, &policyDigest);
        if(policyAlg == TPM_ALG_ERROR)
            continue;
        if(policyList->count < count)
            {
                // If we have not filled up the return list, add this
                // policy to the list;
                policyList->policies[policyList->count].handle = handle;
                policyList->policies[policyList->count].policyHash.hashAlg = policyAlg;
                MemoryCopy(&policyList->policies[policyList->count].policyHash.digest,
                    policyDigest.t.buffer, policyDigest.t.size);
                policyList->count++;
            }
        else
            {
                // If the return list is full but we still have permanent handle
                // available, report this and stop iterating
                more = YES;
                break;
            }
    }
    return more;
```
9.7 IoBuffers.c

9.7.1 Includes and Data Definitions

This definition allows this module to see the values that are private to this module but kept in Global.c for ease of state migration.

```c
#define IO_BUFFER_C
#include "Tpm.h"
#include "IoBuffers_fp.h"
```

9.7.2 Buffers and Functions

These buffers are set aside to hold command and response values. In this implementation, it is not guaranteed that the code will stop accessing the `s_actionInputBuffer` before starting to put values in the `s_actionOutputBuffer` so different buffers are required.

9.7.2.1 MemoryIoBufferAllocationReset()

This function is used to reset the allocation of buffers.

```c
void MemoryIoBufferAllocationReset()
{
    s_actionIoAllocation = 0;
}
```

9.7.2.2 MemoryIoBufferZero()

Function zeros the action I/O buffer at the end of a command. Calling this is not mandatory for proper functionality.

```c
void MemoryIoBufferZero()
{
    memset(s_actionIoBuffer, 0, s_actionIoAllocation);
}
```

9.7.2.3 MemoryGetInBuffer()

This function returns the address of the buffer into which the command parameters will be unmarshaled in preparation for calling the command actions.

```c
BYTE *
MemoryGetInBuffer(
    UINT32 size         // Size, in bytes, required for the input
                      // unmarshaling
)
{
    pAssert(size <= sizeof(s_actionIoBuffer));
    // In this implementation, a static buffer is set aside for the command action
    // buffers. The buffer is shared between input and output. This is because
    // there is no need to allocate for the worst case input and worst case output
```
// at the same time.
// Round size up
#define UoM (sizeof(s_actionIoBuffer[0]))
size = (size + (UoM - 1)) & (UINT32_MAX - (UoM - 1));
memset(s_actionIoBuffer, 0, size);
s_actionIoAllocation = size;
return (BYTE *)&s_actionIoBuffer[0];
}

9.7.2.4 MemoryGetOutBuffer()

This function returns the address of the buffer into which the command action code places its output values.

BYTE *
MemoryGetOutBuffer(
UINT32 size // required size of the buffer
)
{
BYTE        *retVal = (BYTE *)(&s_actionIoBuffer[s_actionIoAllocation / UoM]);
pAssert((size + s_actionIoAllocation) < (sizeof(s_actionIoBuffer)));
// In this implementation, a static buffer is set aside for the command action
// output buffer.
memset(retVal, 0, size);
s_actionIoAllocation += size;
return retVal;
}

9.7.2.5 IsLabelProperlyFormatted()

This function checks that a label is a null-terminated string.

NOTE: this function is here because there was no better place for it.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>string is null terminated</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>string is not null terminated</td>
</tr>
</tbody>
</table>

BOOL
IsLabelProperlyFormatted(
TPM2B *x
)
{
return (((x)->size == 0) || ((x)->buffer[(x)->size - 1] == 0));
}
9.8 Località.c

9.8.1 Includes

```c
#include "Tpm.h"
```

9.8.2 LocalityGetAttributes()

This function will convert a locality expressed as an integer into TPMA_LOCALITY form.

The function returns the locality attribute.

```c
TPMA_LOCALITY
LocalityGetAttributes(
    UINT8 locality     // IN: locality value
)
{
    TPMA_LOCALITY locality_attributes;
    BYTE *localityAsByte = (BYTE *)&locality_attributes;

    MemorySet(&locality_attributes, 0, sizeof(TPMA_LOCALITY));
    switch(locality)
    {
        case 0:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_ZERO);
            break;
        case 1:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_ONE);
            break;
        case 2:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_TWO);
            break;
        case 3:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_THREE);
            break;
        case 4:
            SET_ATTRIBUTE(locality_attributes, TPMA_LOCALITY, TPM_LOC_FOUR);
            break;
        default:
            pAssert(locality > 31);
            *localityAsByte = locality;
            break;
    }
    return locality_attributes;
}
```
9.9 Manufacture.c

9.9.1 Description

This file contains the function that performs the **manufacturing** of the TPM in a simulated environment. These functions should not be used outside of a manufacturing or simulation environment.

9.9.2 Includes and Data Definitions

```c
#define MANUFACTURE_C
#include "Tpm.h"
#include "TpmSizeChecks_fp.h"
```

9.9.3 Functions

9.9.3.1 TPM_CleanSetup()

This function initializes the TPM values in preparation for the TPM's first use. This function will fail if previously called. The TPM can be re-manufactured by calling TPM_Teardown() first and then calling this function again.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>1</td>
<td>manufacturing process previously performed</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT int
TPM_Manufacture(
  int firstTime, // IN: indicates if this is the first call from
  // main()
)
{
  TPM_SU orderlyShutdown;
  
  #if RUNTIME_SIZE_CHECKS
  // Call the function to verify the sizes of values that result from different
  // compile options.
  TpmSizeChecks();
  #endif

  // If TPM has been manufactured, return indication.
  if(!firstTime && g_manufactured)
    return 1;

  // Do power on initializations of the cryptographic libraries.
  CryptInit();

  s_DAPendingOnNV = FALSE;

  // initialize NV
  NvManufacture();

  // Clear the magic value in the DRBG state
  go.drbgState.magic = 0;

  CryptStartup(SU_RESET);

  // default configuration for PCR
  PCRSimStart();
```
 Part 4: Supporting Routines

37 // initialize pre-installed hierarchy data
38 // This should happen after NV is initialized because hierarchy data is
39 // stored in NV.
40 HierarchyPreInstall_Init();
41
42 // initialize dictionary attack parameters
43 DAPreInstall_Init();
44
45 // initialize PP list
46 PhysicalPresencePreInstall_Init();
47
48 // initialize command audit list
49 CommandAuditPreInstall_Init();
50
51 // first start up is required to be Startup(CLEAR)
52 orderlyShutdown = TPM_SU_CLEAR;
53 NV_WRITE_PERSISTENT(orderlyState, orderlyShutdown);
54
55 // initialize the firmware version
56 #ifdef FIRMWARE_V2
57 gp.firmwareV2 = FIRMWARE_V2;
58 #else
59 gp.firmwareV2 = 0;
60 #endif
61 NV_SYNC_PERSISTENT(firmwareV1);
62 NV_SYNC_PERSISTENT(firmwareV2);
63
64 // initialize the total reset counter to 0
65 gp.totalResetCount = 0;
66 NV_SYNC_PERSISTENT(totalResetCount);
67
68 // initialize the clock stuff
69 go.clock = 0;
70 go.clockSafe = YES;
71
72 NvWrite(NV_ORDERLY_DATA, sizeof(ORDERLY_DATA), &go);
73
74 // Commit NV writes. Manufacture process is an artificial process existing
75 // only in simulator environment and it is not defined in the specification
76 // that what should be the expected behavior if the NV write fails at this
77 // point. Therefore, it is assumed the NV write here is always success and
78 // no return code of this function is checked.
79 NvCommit();
80
81 g_manufactured = TRUE;
82
83 return 0;
84 }

9.9.3.2 TPM_TearDown()

This function prepares the TPM for re-manufacture. It should not be implemented in anything other than a simulated TPM.

In this implementation, all that is needs is to stop the cryptographic units and set a flag to indicate that the TPM can be re-manufactured. This should be all that is necessary to start the manufacturing process again.
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>success</td>
</tr>
<tr>
<td>1</td>
<td>TPM not previously manufactured</td>
</tr>
</tbody>
</table>

```
LIB_EXPORT int
TPM_TearDown(
    void
)
{
    g_manufactured = FALSE;
    return 0;
}
```

### 9.9.3.3 TpmEndSimulation()

This function is called at the end of the simulation run. It is used to provoke printing of any statistics that might be needed.

```
LIB_EXPORT void
TpmEndSimulation(
    void
)
{
    #if SIMULATION
    HashLibSimulationEnd();
    SymLibSimulationEnd();
    MathLibSimulationEnd();
    #if ALG_RSA
    RsaSimulationEnd();
    #endif
    #if ALG_ECC
    EccSimulationEnd();
    #endif
    #endif // SIMULATION
}```
9.10 Marshal.c

9.10.1 Introduction

This file contains the marshaling and unmarshaling code.

The marshaling and unmarshaling code and function prototypes are not listed, as the code is repetitive, long, and not very useful to read. Examples of a few unmarshaling routines are provided. Most of the others are similar.

Depending on the table header flags, a type will have an unmarshaling routine and a marshaling routine. The table header flags that control the generation of the unmarshaling and marshaling code are delimited by angle brackets ("<>") in the table header. If no brackets are present, then both unmarshaling and marshaling code is generated (i.e., generation of both marshaling and unmarshaling code is the default).

9.10.2 Unmarshal and Marshal a Value

In TPM 2.0 Part 2, a TPMI_DI_OBJECT is defined by this table:

<table>
<thead>
<tr>
<th>Values</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>{TRANSIENT_FIRST:TRANSIENT_LAST}</td>
<td>allowed range for transient objects</td>
</tr>
<tr>
<td>{PERSISTENT_FIRST:PERSISTENT_LAST}</td>
<td>allowed range for persistent objects</td>
</tr>
<tr>
<td>+TPM_RH_NULL</td>
<td>the null handle</td>
</tr>
<tr>
<td>#TPM_RC_VALUE</td>
<td></td>
</tr>
</tbody>
</table>

This generates the following unmarshaling code:

```
1  TPM_RC
2  TPMI_DH_OBJECT_Unmarshal(TPMI_DH_OBJECT *target, BYTE **buffer, INT32 *size,
3                                  BOOL flag)
4  {
5      TPM_RC    result;
6      result = TPM_HANDLE_Unmarshal((TPM_HANDLE *)target, buffer, size);
7      if(result != TPM_RC_SUCCESS)
8          return result;
9      if(*target == TPM_RH_NULL)
10         {
11            if(flag)
12                return TPM_RC_SUCCESS;
13            else
14                return TPM_RC_VALUE;
15         }
16     if(((*target < TRANSIENT_FIRST) || (*target > TRANSIENT_LAST))
17        &&((*target < PERSISTENT_FIRST) || (*target > PERSISTENT_LAST)))
18         return TPM_RC_VALUE;
19     return TPM_RC_SUCCESS;
20 }
```

and the following marshaling code:

```
1  UINT16
2  TPMI_DH_OBJECT_Marshal(TPMI_DH_OBJECT *source, BYTE **buffer, INT32 *size)
```
An additional script is used to do the work that might be done by a linker or globally optimizing compiler. It searches for functions like `TPMI_DH_OBJECT_Marshal()` that do nothing but call another function and replaces the function with a `#define`.

```c
#define TPMI_DH_OBJECT_Marshal(source, buffer, size)   \
    UINT32_Marshal((UINT32 *)source, buffer, size)
```

When replacing the function with a `#define`, the `#define` is placed in `marshal_fp.h` and the function body is removed from `marshal.c`.

### 9.10.3 Unmarshal and Marshal a Union

In TPM 2.0 Part 2, a `TPMU_PUBLIC_PARMS` union is defined by:

**Table xxx — Definition of `TPMU_PUBLIC_PARMS` Union `<IN/OUT, S>`**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Selector</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>keyedHash</td>
<td><code>TPMS_KEYEDHASH_PARMS</code></td>
<td><code>TPM_ALG_KEYEDHASH</code></td>
<td>sign</td>
</tr>
<tr>
<td>symDetail</td>
<td><code>TPMT_SYM_DEF_OBJECT</code></td>
<td><code>TPM_ALG_SYMCIPHER</code></td>
<td>a symmetric block cipher</td>
</tr>
<tr>
<td>rsaDetail</td>
<td><code>TPMS_RSA_PARMS</code></td>
<td><code>TPM_ALG_RSA</code></td>
<td>decrypt + sign</td>
</tr>
<tr>
<td>eccDetail</td>
<td><code>TPMS_ECC_PARMS</code></td>
<td><code>TPM_ALG_ECC</code></td>
<td>decrypt + sign</td>
</tr>
<tr>
<td>asymDetail</td>
<td><code>TPMS ASYM_PARMS</code></td>
<td></td>
<td>common scheme structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>for RSA and ECC keys</td>
</tr>
</tbody>
</table>

NOTE The Description column indicates which of `TPMA_OBJECT.decrypt` or `TPMA_OBJECT.sign` may be set. "+" indicates that both may be set but one shall be set. "|" indicates the optional settings.

From this table, the following unmarshaling code is generated.

```c
TPM_RC TPMU_PUBLIC_PARMS_Unmarshal(TPMU_PUBLIC_PARMS *target, BYTE **buffer, INT32 *size, 
                                   UINT32 selector)
{
    switch(selector) {
    #ifdef TPM_ALG_KEYEDHASH
      case TPM_ALG_KEYEDHASH:
        return TPMS_KEYEDHASH_PARMS_Unmarshal((TPMS_KEYEDHASH_PARMS *)&(target->keyedHash), buffer, size);
        #endif
    #ifdef TPM_ALG_SYMCIPHER
      case TPM_ALG_SYMCIPHER:
        return TPMT_SYM_DEF_OBJECT_Unmarshal((TPMT_SYM_DEF_OBJECT *)&(target->symDetail), buffer, size, FALSE);
        #endif
    #ifdef TPM_ALG_RSA
      case TPM_ALG_RSA:
        return TPMS_RSA_PARMS_Unmarshal((TPMS_RSA_PARMS *)&(target->rsaDetail), buffer, size);
        #endif
    #ifdef TPM_ALG_ECC
      case TPM_ALG_ECC:
        return TPMS_ECC_PARMS_Unmarshal((TPMS_ECC_PARMS *)&(target->eccDetail), buffer, size);
        #endif
    }
```
NOTE  The #ifdef/#endif directives are added whenever a value is dependent on an algorithm ID so that removing the algorithm definition will remove the related code.

The marshaling code for the union is:

```
UINT16
TPMU_PUBLIC_PARMS_Marshal(TPMU_PUBLIC_PARMS *source, BYTE **buffer, INT32 *size,
UINT32 selector)
{
    switch(selector) {
        #ifdef TPM_ALG_KEYEDHASH
        case TPM_ALG_KEYEDHASH:
            return TPMS_KEYEDHASH_PARMS_Marshal(
                (TPMS_KEYEDHASH_PARMS *)&(source->keyedHash), buffer, size);
        #endif

        #ifdef TPM_ALG_SYMCIPHER
        case TPM_ALG_SYMCIPHER:
            return TPMT_SYM_DEF_OBJECT_Marshal(
                (TPMT_SYM_DEF_OBJECT *)&(source->symDetail), buffer, size);
        #endif

        #ifdef TPM_ALG_RSA
        case TPM_ALG_RSA:
            return TPMS_RSA_PARMS_Marshal(
                (TPMS_RSA_PARMS *)&(source->rsaDetail), buffer, size);
        #endif

        #ifdef TPM_ALG_ECC
        case TPM_ALG_ECC:
            return TPMS_ECC_PARMS_Marshal(
                (TPMS_ECC_PARMS *)&(source->eccDetail), buffer, size);
        #endif
        }
    assert(1);
    return 0;
}
```

For the marshaling and unmarshaling code, a value in the structure containing the union provides the value used for selector. The example in the next section illustrates this.
9.10.4 Unmarshal and Marshal a Structure

In TPM 2.0 Part 2, the TPMT_PUBLIC structure is defined by:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>TPMI_ALG_PUBLIC</td>
<td>&quot;algorithm&quot; associated with this object</td>
</tr>
<tr>
<td>nameAlg</td>
<td>+TPMI_ALG_HASH</td>
<td>algorithm used for computing the Name of the object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE The &quot;+&quot; indicates that the instance of a TPMT_PUBLIC may have</td>
</tr>
<tr>
<td></td>
<td></td>
<td>a &quot;+&quot; to indicate that the nameAlg may be TPM_ALG_NULL.</td>
</tr>
<tr>
<td>objectAttributes</td>
<td>TPMA_OBJECT</td>
<td>attributes that, along with type, determine the manipulations of this object</td>
</tr>
<tr>
<td>authPolicy</td>
<td>TPM2B_DIGEST</td>
<td>optional policy for using this key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The policy is computed using the nameAlg of the object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE shall be the Empty Buffer if no authorization policy is present</td>
</tr>
<tr>
<td>[type]parameters</td>
<td>TPMU_PUBLIC_PARMS</td>
<td>the algorithm or structure details</td>
</tr>
<tr>
<td>[type]unique</td>
<td>TPMU_PUBLIC_ID</td>
<td>the unique identifier of the structure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For an asymmetric key, this would be the public key.</td>
</tr>
</tbody>
</table>

This structure is tagged (the first value indicates the structure type), and that tag is used to determine how the parameters and unique fields are unmarshaled and marshaled. The use of the type for specifying the union selector is emphasized below.

The unmarshaling code for the structure in the table above is:

```c
TPM_RC
TPMT_PUBLIC_Unmarshal(TPMT_PUBLIC *target, BYTE **buffer, INT32 *size, BOOL flag)
{
  TPM_RC   result;
  result = TPMI_ALG_PUBLIC_Unmarshal((TPMI_ALG_PUBLIC *)(target->type),
                                    buffer, size);
  if(result != TPM_RC_SUCCESS)
    return result;
  result = TPMI_ALG_HASH_Unmarshal((TPMI_ALG_HASH *)(target->nameAlg),
                                    buffer, size, flag);
  if(result != TPM_RC_SUCCESS)
    return result;
  result = TPMA_OBJECT_Unmarshal((TPMA_OBJECT *)(target->objectAttributes),
                                  buffer, size);
  if(result != TPM_RC_SUCCESS)
    return result;
  result = TPM2B_DIGEST_Unmarshal((TPM2B_DIGEST *)(target->authPolicy),
                                  buffer, size);
  if(result != TPM_RC_SUCCESS)
    return result;
  result = TPMU_PUBLIC_PARMS_Unmarshal((TPMU_PUBLIC_PARMS *)(target->parameters),
                                        buffer, size, (UINT32)target->type);
  if(result != TPM_RC_SUCCESS)
    return result;
  result = TPMU_PUBLIC_ID_Unmarshal((TPMU_PUBLIC_ID *)(target->unique),
                                      buffer, size, (UINT32)target->type);
  if(result != TPM_RC_SUCCESS)
    return result;
  return TPM_RC_SUCCESS;
}
```
The marshaling code for the TPMT_PUBLIC structure is:

```c
UINT16
TPMT_PUBLIC_Marshal(TPMT_PUBLIC *source, BYTE **buffer, INT32 *size)
{
    UINT16 result = 0;
    result = (UINT16)(result + TPMI_ALG_PUBLIC_Marshal(
                  (TPMI_ALG_PUBLIC *)&(source->type), buffer, size));
    result = (UINT16)(result + TPMI_ALG_HASH_Marshal(
                  (TPMI_ALG_HASH *)&(source->nameAlg), buffer, size));
    result = (UINT16)(result + TPM_OBJECT_Marshal(
                  (TPMA_OBJECT *)&(source->objectAttributes), buffer, size));
    result = (UINT16)(result + TPM2B_DIGEST_Marshal(
                  (TPM2B_DIGEST *)&(source->authPolicy), buffer, size));
    result = (UINT16)(result + TPMU_PUBLIC_PARMS_Marshal(
                  (TPMU_PUBLIC_PARMS *)&(source->parameters), buffer, size,
                  (UINT32)&(source->type)));
    result = (UINT16)(result + TPMU_PUBLIC_ID_Marshal(
                  (TPMU_PUBLIC_ID *)&(source->unique), buffer, size,
                  (UINT32)&(source->type)));
    return result;
}
```

### 9.10.5 Unmarshal and Marshal an Array

In TPM 2.0 Part 2, the TPML_DIGEST is defined by:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>count {2:}</td>
<td>UINT32</td>
<td>number of digests in the list, minimum is two</td>
</tr>
<tr>
<td>digests[count][8]</td>
<td>TPM2B_DIGEST</td>
<td>a list of digests</td>
</tr>
<tr>
<td>#TPM_RC_SIZE</td>
<td></td>
<td>response code when count is not at least two or is greater than 8</td>
</tr>
</tbody>
</table>

The `digests` parameter is an array of up to `count` structures (TPM2B_DIGESTS). The auto-generated code to Unmarshal this structure is:

```c
TPM_RC
TPML_DIGEST_Unmarshal(TPML_DIGEST *target, BYTE **buffer, INT32 *size)
{
    TPM_RC result;
    result = UINT32_Unmarshal((UINT32 *)&(target->count), buffer, size);
    if(result != TPM_RC_SUCCESS)
        return result;

    if( (target->count < 2)) // This check is triggered by the {2:} notation
        return TPM_RC_SIZE; // on 'count'

    if((target->count) > 8)  // This check is triggered by the {:8} notation
        return TPM_RC_SIZE;
```
The routine unmarshals a `count` value and passes that value to a routine that unmarshals an array of `TPM2B_DIGEST` values. The unmarshaling code for the array is:

```c
TPM_RC
TPM2B_DIGEST_Array.Unmarshal(TPM2B_DIGEST *target, BYTE **buffer, INT32 *size,
                             INT32 count)
{
    TPM_RC result;
    INT32 i;
    for(i = 0; i < count; i++) {
        result = TPM2B_DIGEST_Unmarshal(&target[i], buffer, size);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    return TPM_RC_SUCCESS;
}
```

Marshaling of the `TPML_DIGEST` uses a similar scheme with a structure specifying the number of elements in an array and a subsequent call to a routine to marshal an array of that type.

```c
UINT16
TPML_DIGEST_Marshal(TPML_DIGEST *source, BYTE **buffer, INT32 *size)
{
    UINT16 result = 0;
    result = (UINT16)(result + UINT32_Marshal((UINT32 *)&(source->count), buffer,
                                               size));
    result = (UINT16)(result + TPM2B_DIGEST_Array_Marshal(
                        (TPM2B_DIGEST *)(source->digests), buffer, size,
                        (INT32)(source->count)));
    return result;
}
```

The marshaling code for the array is:

```c
TPM_RC
TPM2B_DIGEST_Array_Unmarshal(TPM2B_DIGEST *target, BYTE **buffer, INT32 *size,
                              INT32 count)
{
    TPM_RC result;
    INT32 i;
    for(i = 0; i < count; i++) {
        result = TPM2B_DIGEST_Unmarshal(&target[i], buffer, size);
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    return TPM_RC_SUCCESS;
}
```
9.10.6 TPM2B Handling

A TPM2B structure is handled as a special case. The unmarshaling code is similar to what is shown in 9.10.5 but the unmarshaling/marshaling is to a union element. Each TPM2B is a union of two sized buffers, one of which is type specific (the ‘t’ element) and the other is a generic value (the ‘b’ element). This allows each of the TPM2B structures to have some inheritance property with all other TPM2B. The purpose is to allow functions that have parameters that can be any TPM2B structure while allowing other functions to be specific about the type of the TPM2B that is used. When the generic structure is allowed, the input parameter would use the ‘b’ element and when the type-specific structure is required, the ‘t’ element is used.

When marshaling a TPM2B where the second member is a BYTE array, the size parameter indicates the size of the array. The second member can also be a structure. In this case, the caller does not prefill the size member. The marshaling code must marshal the structure and then back fill the calculated size.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>UINT16</td>
<td>Size of the operand</td>
</tr>
<tr>
<td>buffer [size]</td>
<td>BYTE</td>
<td>The operand</td>
</tr>
</tbody>
</table>

```c
TPM2B EVENT_Unmarshal(TPM2B EVENT *target, BYTE **buffer, INT32 *size)
{
    TPM_RC    result;
    result = UINT16_Unmarshal((UINT16 *)&(target->t.size), buffer, size);
    if(result != TPM_RC_SUCCESS)
        return result;
    // if size equal to 0, the rest of the structure is a zero buffer
    // so stop processing
    if((target->t.size) == 0)
        return TPM_RC_SUCCESS;
    if((target->t.size) > 1024) // This check is triggered by the {:1024}
        // notation on ‘buffer’
        return TPM_RC_SIZE;
    result = BYTE_Array_Unmarshal((BYTE *)(target->t.buffer), buffer, size,
                                 (INT32)(target->t.size));
    if(result != TPM_RC_SUCCESS)
        return result;
    return TPM_RC_SUCCESS;
}
```

typedef union {
    struct {
        UINT16        size;
        BYTE          buffer[1024];
    }            t;
    TPM2B        b;
} TPM2B EVENT;
9.11 MathOnByteBuffers.c

9.11.1 Introduction

This file contains implementation of the math functions that are performed with canonical integers in byte buffers. The canonical integer is big-endian bytes.

```c
#include "Tpm.h"
```

9.11.2 Functions

9.11.2.1 UnsignedCmpB

This function compares two unsigned values. The values are byte-aligned, big-endian numbers (e.g., a hash).

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if (a &gt; b)</td>
</tr>
<tr>
<td>0</td>
<td>if (a = b)</td>
</tr>
<tr>
<td>-1</td>
<td>if (a &lt; b)</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT int UnsignedCompareB(
    UINT32 aSize,      // IN: size of a
    const BYTE *a,     // IN: a
    UINT32 bSize,      // IN: size of b
    const BYTE *b      // IN: b
)
{
    UINT32 i;
    if (aSize > bSize)
        return 1;
    else if (aSize < bSize)
        return -1;
    else
    {
        for (i = 0; i < aSize; i++)
            if (a[i] != b[i])
                return (a[i] > b[i]) ? 1 : -1;
    }
    return 0;
}
```

9.11.2.2 SignedCompareB()

Compare two signed integers:
9.11.2.3 ModExpB

This function is used to do modular exponentiation in support of RSA. The most typical uses are: \( c = m^e \mod n \) (RSA encrypt) and \( m = c^d \mod n \) (RSA decrypt). When doing decryption, the \( e \) parameter of the function will contain the private exponent \( d \) instead of the public exponent \( e \).

If the results will not fit in the provided buffer, an error is returned (CRYPT_ERROR_UNDERFLOW). If the results is smaller than the buffer, the results is de-normalized.

This version is intended for use with RSA and requires that \( m \) be less than \( n \).

---

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>if ( a &gt; b )</td>
</tr>
<tr>
<td>0</td>
<td>if ( a = b )</td>
</tr>
<tr>
<td>-1</td>
<td>if ( a &lt; b )</td>
</tr>
</tbody>
</table>

```c
int SignedCompareB(
    const UINT32 aSize,  // IN: size of a
    const BYTE *a,       // IN: a buffer
    const UINT32 bSize,  // IN: size of b
    const BYTE *b        // IN: b buffer
) {
    int signA, signB;    // sign of a and b

    // For positive or 0, sign_a is 1
    // for negative, sign_a is 0
    signA = ((a[0] & 0x80) == 0) ? 1 : 0;

    // For positive or 0, sign_b is 1
    // for negative, sign_b is 0
    signB = ((b[0] & 0x80) == 0) ? 1 : 0;

    if(signA != signB)
        return signA - signB;
    if(signA == 1)
        // do unsigned compare function
        return UnsignedCompareB(aSize, a, bSize, b);
    else
        // do unsigned compare the other way
        return 0 - UnsignedCompareB(aSize, a, bSize, b);
}
```
const UINT32  eSize,
const BYTE    *e,       // IN: power
const UINT32  nSize,   // IN: modulus
const BYTE    *n
)
{
    BN_MAX(bnC);
    BN_MAX(bnM);
    BN_MAX(bnE);
    BN_MAX(bnN);
    NUMBYTES         tSize = (NUMBYTES)nSize;
    TPM_RC           retVal = TPM_RC_SUCCESS;
    // Convert input parameters
    BnFromBytes(bnM, m, (NUMBYTES)mSize);
    BnFromBytes(bnE, e, (NUMBYTES)eSize);
    BnFromBytes(bnN, n, (NUMBYTES)nSize);

    // Make sure that the output is big enough to hold the result
    // and that 'm' is less than 'n' (the modulus)
    if(cSize < nSize)
        ERROR_RETURN(TPM_RC_NO_RESULT);
    if(BnUnsignedCmp(bnM, bnN) >= 0)
        ERROR_RETURN(TPM_RC_SIZE);
    BnModExp(bnC, bnM, bnE, bnN);
    BnToBytes(bnC, c, &tSize);
    Exit:
    return retVal;
}

9.11.2.4 DivideB()

Divide an integer (n) by an integer (d) producing a quotient (q) and a remainder (r). If q or r is not needed, then the pointer to them may be set to NULL.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>q or r is too small to receive the result</td>
</tr>
</tbody>
</table>

LIB_EXPORT TPM_RC DivideB(
    const TPM2B     *n,       // IN: numerator
    const TPM2B     *d,       // IN: denominator
    TPM2B           *q,       // OUT: quotient
    TPM2B           *r        // OUT: remainder
)
{
    BN_MAX_INITIALIZED(bnN, n);
    BN_MAX_INITIALIZED(bnD, d);
    BN_MAX(bnQ);
    BN_MAX(bnR);
    // Do divide with converted values
    BnDiv(bnQ, bnR, bnN, bnD);
    // Convert the BIGNUM result back to 2B format using the size of the original number
    if(q != NULL)
        if(!BnTo2B(bnQ, q, q->size))
            return TPM_RC_NO_RESULT;
    if(r != NULL)
        if(!BnTo2B(bnR, r, r->size))
            return TPM_RC_NO_RESULT;
    return TPM_RC_SUCCESS;
9.11.2.5 AdjustNumberB()

Remove/add leading zeros from a number in a TPM2B. Will try to make the number by adding or removing leading zeros. If the number is larger than the requested size, it will make the number as small as possible. Setting requestedSize to zero is equivalent to requesting that the number be normalized.

```c
9.11.2.5 AdjustNumberB()

   UINT16
   AdjustNumberB(
      TPM2B       *num,
      UINT16       requestedSize
   )
{
   BYTE            *from;
   UINT16           i;
   // See if number is already the requested size
   if(num->size == requestedSize)
      return requestedSize;
   from = num->buffer;
   if (num->size > requestedSize)
   {
      // This is a request to shift the number to the left (remove leading zeros)
      // Find the first non-zero byte. Don't look past the point where removing
      // more zeros would make the number smaller than requested, and don't throw
      // away any significant digits.
      for(i = num->size; *from == 0 && i > requestedSize; from++, i--);
      if(i < num->size)
      {
         num->size = i;
         MemoryCopy(num->buffer, from, i);
      }
   }
   // This is a request to shift the number to the right (add leading zeros)
   else
   {
      MemoryCopy(&num->buffer[requestedSize - num->size], num->buffer, num->size);
      MemorySet(num->buffer, 0, requestedSize- num->size);
      num->size = requestedSize;
   }
   return num->size;
}
```

9.11.2.6 ShiftLeft()

This function shifts a byte buffer (a TPM2B) one byte to the left. That is, the most significant bit of the most significant byte is lost.

```c
9.11.2.6 ShiftLeft()

   TPM2B *
   ShiftLeft(
      TPM2B       *value       // IN/OUT: value to shift and shifted value out
   )
{
   UINT16       count = value->size;
   BYTE            *buffer = value->buffer;
   if(count > 0)
   {
      for(count -= 1; count > 0; buffer++, count--)
      {
         buffer[0] = (buffer[0] << 1) + ((buffer[1] & 0x80) ? 1 : 0);
      }
      *buffer <<= 1;
   }
```
9.11.2.7 IsNumeric()

Verifies that all the characters are simple numeric (0-9)

```c
BOOL IsNumeric(TPM2B *value) {
    UINT16 i;
    for(i = 0; i < value->size; i++) {
        if(value->buffer[i] < '0' || value->buffer[i] > '9')
            return FALSE;
    }
    return TRUE;
}
```
9.12 Memory.c

9.12.1 Description

This file contains a set of miscellaneous memory manipulation routines. Many of the functions have the same semantics as functions defined in string.h. Those functions are not used directly in the TPM because they are not safe.

This version uses string.h after adding guards. This is because the math libraries invariably use those functions so it is not practical to prevent those library functions from being pulled into the build.

9.12.2 Includes and Data Definitions

```c
#include "Tpm.h"
#include "Memory_fp.h"
```

9.12.3 Functions

9.12.3.1 MemoryCopy()

This is an alias for memmove. This is used in place of memcpy because some of the moves may overlap and rather than try to make sure that memmove is used when necessary, it is always used. The #if 0 is used to prevent instantiation of the MemoryCopy() function so that the #define is always used.

```c
void MemoryCopy(
    void *dest,
    const void *src,
    int sSize
){
    memmove(dest, src, sSize);
}
```

9.12.3.2 MemoryEqual()

This function indicates if two buffers have the same values in the indicated number of bytes.

```
BOOL MemoryEqual(
    const void *buffer1,       // IN: compare buffer1
    const void *buffer2,       // IN: compare buffer2
    unsigned int size          // IN: size of bytes being compared
){
    BYTE         equal = 0;
    const BYTE  *b1 = (BYTE *)buffer1;
    const BYTE  *b2 = (BYTE *)buffer2;
    // Compare all bytes so that there is no leakage of information
    // due to timing differences.
    for(; size > 0; size--)
```

```c
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>all octets are the same</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>all octets are not the same</td>
</tr>
</tbody>
</table>
```
equal |= (*b1++ ^ *b2++);
return (equal == 0);
}

9.12.3.3 MemoryCopy2B()

This function copies a TPM2B. This can be used when the TPM2B types are the same or different.
This function returns the number of octets in the data buffer of the TPM2B.

LIB_EXPORT INT16
MemoryCopy2B(
TPM2B *dest, // OUT: receiving TPM2B
const TPM2B *source, // IN: source TPM2B
unsigned int dSize // IN: size of the receiving buffer
)
{
    pAssert(dest != NULL);
    if(source == NULL)
        dest->size = 0;
    else
    {
        pAssert(source->size <= dSize);
        MemoryCopy(dest->buffer, source->buffer, source->size);
        dest->size = source->size;
    }
    return dest->size;
}

9.12.3.4 MemoryConcat2B()

This function will concatenate the buffer contents of a TPM2B to an the buffer contents of another TPM2B
and adjust the size accordingly (a := (a | b)).

void
MemoryConcat2B(
TPM2B *aInOut, // IN/OUT: destination 2B
TPM2B *bIn, // IN: second 2B
unsigned int aMaxSize // IN: The size of aInOut.buffer (max values for
// aInOut.size)
)
{
    pAssert(bIn->size <= aMaxSize - aInOut->size);
    MemoryCopy(&aInOut->buffer[aInOut->size], &bIn->buffer, bIn->size);
    aInOut->size = aInOut->size + bIn->size;
    return;
}

9.12.3.5 MemoryEqual2B()

This function will compare two TPM2B structures. To be equal, they need to be the same size and the
buffer contexts need to be the same in all octets.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>size and buffer contents are the same</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>size or buffer contents are not the same</td>
</tr>
</tbody>
</table>

BOOL
MemoryEqual2B();
const TPM2B *aIn, // IN: compare value
const TPM2B *bIn    // IN: compare value

if(aIn->size != bIn->size)
    return FALSE;
return MemoryEqual(aIn->buffer, bIn->buffer, aIn->size);

9.12.3.6 MemorySet()

This function will set all the octets in the specified memory range to the specified octet value.

NOTE: A previous version had an additional parameter (dSize) that was intended to make sure that the destination would not be overrun. The problem is that, in use, all that was happening was that the value of size was used for dSize so there was no benefit in the extra parameter.

void MemorySet(void *dest, int value, size_t size)
{
    memset(dest, value, size);
}

9.12.3.7 MemoryPad2B()

Function to pad a TPM2B with zeros and adjust the size.

void MemoryPad2B(TPM2B *b, UINT16 newSize)
{
    MemorySet(&b->buffer[b->size], 0, newSize - b->size);
    b->size = newSize;
}

9.12.3.8 Uint16ToByteArray()

Function to write an integer to a byte array

void Uint16ToByteArray(UINT16 i, BYTE *a)
{
    a[1] = (BYTE)(i); i >>= 8;
    a[0] = (BYTE)(i);
}

9.12.3.9 Uint32ToByteArray()

Function to write an integer to a byte array

void
98  Uint32ToByteArray(
99    UINT32 i,
100    BYTE *a
101  )
102  {
103    a[3] = (BYTE)(i); i >>= 8;
104    a[2] = (BYTE)(i); i >>= 8;
105    a[1] = (BYTE)(i); i >>= 8;
106    a[0] = (BYTE)(i);
107  }

9.12.3.10  Uint64ToByteArray()

Function to write an integer to a byte array

108  void
109  Uint64ToByteArray(
110    UINT64 i,
111    BYTE *a
112  )
113  {
114    a[7] = (BYTE)(i); i >>= 8;
115    a[6] = (BYTE)(i); i >>= 8;
116    a[5] = (BYTE)(i); i >>= 8;
117    a[4] = (BYTE)(i); i >>= 8;
118    a[3] = (BYTE)(i); i >>= 8;
119    a[2] = (BYTE)(i); i >>= 8;
120    a[1] = (BYTE)(i); i >>= 8;
121    a[0] = (BYTE)(i);
122  }

9.12.3.11  ByteArrayToUint16()

Function to write an integer to a byte array

123  UINT16
124  ByteArrayToUint16(
125    BYTE *a
126  )
127  {
128    UINT16 retVal;
129    retVal = a[0]; retVal <<= 8;
130    retVal += a[1];
131    return retVal;
132  }

9.12.3.12  ByteArrayToUint32()

Function to write an integer to a byte array

133  UINT32
134  ByteArrayToUint32(
135    BYTE *a
136  )
137  {
138    UINT32 retVal;
139    retVal = a[0]; retVal <<= 8;
140    retVal += a[1]; retVal <<= 8;
141    retVal += a[2]; retVal <<= 8;
142    return retVal;
9.12.3.13 ByteArrayToUint64()

Function to write an integer to a byte array

```
UINT64 ByteArrayToUint64(
    BYTE *a
)
{
    UINT64 retVal;
    retVal = a[0]; retVal <<= 8;
    retVal += a[1]; retVal <<= 8;
    retVal += a[2]; retVal <<= 8;
    retVal += a[3]; retVal <<= 8;
    retVal += a[4]; retVal <<= 8;
    retVal += a[5]; retVal <<= 8;
    retVal += a[6]; retVal <<= 8;
    return retVal;
}
```
9.13 Power.c

9.13.1 Description

This file contains functions that receive the simulated power state transitions of the TPM.

9.13.2 Includes and Data Definitions

```c
#define POWER_C
#include "Tpm.h"
```

9.13.3 Functions

9.13.3.1 TPMInit()

This function is used to process a power on event.

```c
void TPMInit(
  void
)
{
  // Set state as not initialized. This means that Startup is required
  g_initialized = FALSE;
  return;
}
```

9.13.3.2 TPMRegisterStartup()

This function registers the fact that the TPM has been initialized (a TPM2_Startup() has completed successfully).

```c
void TPMRegisterStartup(
  void
)
{
  g_initialized = TRUE;
  return;
}
```

9.13.3.3 TPMIsStarted()

Indicates if the TPM has been initialized (a TPM2_Startup() has completed successfully after a _TPM_Init()).

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>TPM has been initialized</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>TPM has not been initialized</td>
</tr>
</tbody>
</table>

```c
BOOL TPMIsStarted(
  void
)
{
}
```
return g_initialized;
9.14 PropertyCap.c

9.14.1 Description

This file contains the functions that are used for accessing the TPM_CAP_TPMPROPERTY values.

9.14.2 Includes

```c
#include "Tpm.h"
```

9.14.3 Functions

9.14.3.1 TPMPropertyIsDefined()

This function accepts a property selection and, if so, sets value to the value of the property.

All the fixed values are vendor dependent or determined by a platform-specific specification. The values in the table below are examples and should be changed by the vendor.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>referenced property exists and value set</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>referenced property does not exist</td>
</tr>
</tbody>
</table>

```c
static BOOL TPMPropertyIsDefined(
    TPM_PT property,     // IN: property
    UINT32 *value        // OUT: property value
) {
    switch (property)
    {
    case TPM_PT_FAMILY_INDICATOR:
        // from the title page of the specification
        // For this specification, the value is "2.0".
        *value = TPM_SPEC_FAMILY;
        break;
    case TPM_PT_LEVEL:
        // from the title page of the specification
        *value = TPM_SPEC_LEVEL;
        break;
    case TPM_PT_REVISION:
        // from the title page of the specification
        *value = TPM_SPEC_VERSION;
        break;
    case TPM_PT_DAY_OF_YEAR:
        // computed from the date value on the title page of the specification
        *value = TPM_SPEC_DAY_OF_YEAR;
        break;
    case TPM_PT_YEAR:
        // from the title page of the specification
        *value = TPM_SPEC_YEAR;
        break;
    case TPM_PT_MANUFACTURER:
        // vendor ID unique to each TPM manufacturer
        *value = BYTE_ARRAY_TO_UINT32(MANUFACTURER);
        break;
    case TPM_PT_VENDOR_STRING_1:
        // first four characters of the vendor ID string
        *value = BYTE_ARRAY_TO_UINT32(VENDOR_STRING_1);
    ```
break;

case TPM_PT_VENDOR_STRING_2:
  // second four characters of the vendor ID string
  #ifdef VENDOR_STRING_2
    *value = BYTE_ARRAY_TO_UINT32(VENDOR_STRING_2);
  #else
    *value = 0;
  #endif
  break;

case TPM_PT_VENDOR_STRING_3:
  // third four characters of the vendor ID string
  #ifdef VENDOR_STRING_3
    *value = BYTE_ARRAY_TO_UINT32(VENDOR_STRING_3);
  #else
    *value = 0;
  #endif
  break;

case TPM_PT_VENDOR_STRING_4:
  // fourth four characters of the vendor ID string
  #ifdef VENDOR_STRING_4
    *value = BYTE_ARRAY_TO_UINT32(VENDOR_STRING_4);
  #else
    *value = 0;
  #endif
  break;

case TPM_PT_VENDOR_TPM_TYPE:
  // vendor-defined value indicating the TPM model
  *value = 1;
  break;

case TPM_PT_FIRMWARE_VERSION_1:
  // more significant 32-bits of a vendor-specific value
  *value = gp.firmwareV1;
  break;

case TPM_PT_FIRMWARE_VERSION_2:
  // less significant 32-bits of a vendor-specific value
  *value = gp.firmwareV2;
  break;

case TPM_PT_INPUT_BUFFER:
  // maximum size of TPM2B_MAX_BUFFER
  *value = MAX_DIGEST_BUFFER;
  break;

case TPM_PT_HR_TRANSIENT_MIN:
  // minimum number of transient objects that can be held in TPM
  // RAM
  *value = MAX_LOADED_OBJECTS;
  break;

case TPM_PT_HR_PERSISTENT_MIN:
  // minimum number of persistent objects that can be held in
  // TPM NV memory
  // In this implementation, there is no minimum number of
  // persistent objects.
  *value = MIN_EVICT_OBJECTS;
  break;

case TPM_PT_HR_LOADED_MIN:
  // minimum number of authorization sessions that can be held in
  // TPM RAM
  *value = MAX_LOADED_SESSIONS;
  break;

case TPM_PT_ACTIVE_SESSIONS_MAX:
  // number of authorization sessions that may be active at a time
  *value = MAX_ACTIVE_SESSIONS;
  break;

case TPM_PT_PCR_COUNT:
  // number of PCR implemented
  *value = IMPLEMENTATION_PCR;
  break;
case TPM_PT_PCR_SELECT_MIN:
    // minimum number of bytes in a TPMS_PCR_SELECT.sizeOfSelect
    *value = PCR_SELECT_MIN;
    break;

case TPM_PT_CONTEXT_GAP_MAX:
    // maximum allowed difference (unsigned) between the contextID
    // values of two saved session contexts
    *value = ((UINT32)1 << (sizeof(CONTEXT_SLOT) * 8)) - 1;
    break;

case TPM_PT_NV_COUNTERS_MAX:
    // maximum number of NV indexes that are allowed to have the
    // TPMA_NV_COUNTER attribute SET
    // In this implementation, there is no limitation on the number
    // of counters, except for the size of the NV Index memory.
    *value = 0;
    break;

case TPM_PT_NV_INDEX_MAX:
    // maximum size of an NV index data area
    *value = MAX_NV_INDEX_SIZE;
    break;

case TPM_PT_MEMORY:
    // a TPMA_MEMORY indicating the memory management method for the TPM
    {
        union
        {
            TPMA_MEMORY att;
            UINT32 u32;
        } attributes = {{0}};
        SET_ATTRIBUTE(attributes.att, TPMA_MEMORY, sharedNV);
        SET_ATTRIBUTE(attributes.att, TPMA_MEMORY, objectCopiedToRam);
        // Note: For a LSb0 machine, the bits in a bit field are in the correct
        // order even if the machine is MSB0. For a MSb0 machine, a TPMA will
        // be an integer manipulated by masking (USE_BIT_FIELD_STRUCTURES will
        // be NO) so the bits are manipulate correctly.
        *value = attributes.u32;
    }
    break;

case TPM_PT_CLOCK_UPDATE:
    // interval, In seconds, between updates to the copy of
    // TPMS_TIME_INFO .clock in NV
    *value = (1 << NV_CLOCK_UPDATE_INTERVAL);
    break;

case TPM_PT_CONTEXT_HASH:
    // algorithm used for the integrity hash on saved contexts and
    // for digesting the fuData of TPM2_FirmwareRead()
    *value = CONTEXT_INTEGRITY_HASH_ALG;
    break;

case TPM_PT_CONTEXT_SYM:
    // algorithm used for encryption of saved contexts
    *value = CONTEXT_ENCRYPT_ALG;
    break;

case TPM_PT_CONTEXT_SYM_SIZE:
    // size of the key used for encryption of saved contexts
    *value = CONTEXT_ENCRYPT_KEY_BITS;
    break;

case TPM_PT_ORDERLY_COUNT:
    // maximum difference between the volatile and non-volatile
    // versions of TPMA_NV_COUNTER that have TPMA_NV_ORDERLY SET
    *value = MAX_ORDERLY_COUNT;
    break;

case TPM_PT_MAX_COMMAND_SIZE:
    // maximum value for 'commandSize'
    *value = MAX_COMMAND_SIZE;
    break;

case TPM_PT_MAX_RESPONSE_SIZE:
170  // maximum value for 'responseSize'
171  *value = MAX_RESPONSE_SIZE;
172  break;
173  case TPM_PT_MAX_DIGEST:
174  // maximum size of a digest that can be produced by the TPM
175  *value = sizeof(TPMU_HA);
176  break;
177  case TPM_PT_MAX_OBJECT_CONTEXT:
178  // Header has 'sequence', 'handle' and 'hierarchy'
179  #define SIZE_OF_CONTEXT_HEADER
180       sizeof(UINT64) + sizeof(TPMI_DH_CONTEXT) + sizeof(TPMI RH HIERARCHY)
181  #define SIZE_OFCONTEXT_INTEGRITY (sizeof(UINT16) + CONTEXT_INTEGRITY_HASH_SIZE)
182  #define SIZE_OF_FINGERPRINT sizeof(UINT64)
183  #define SIZE_OF_CTX BLOB_OVERHEAD
184       (sizeof(UINT16) + SIZE_OF_CONTEXT_INTEGRITY + SIZE_OF_FINGERPRINT)
185  #define SIZE_OF_CTX OVERHEAD
186       (SIZE_OFCONTEXT_HEADER + SIZE_OF_CTX BLOB_OVERHEAD)
187  #if 0
188  // maximum size of a TPMS_CONTEXT that will be returned by
189  // TPM2_ContextSave for object context
190  *value = 0;
191  // adding sequence, saved handle and hierarchy
192  *value += sizeof(UINT64) + sizeof(TPMI_DH_CONTEXT) +
193       sizeof(TPMI RH HIERARCHY);
194  // Add size field in TPM2B_CONTEXT
195  *value += sizeof(UINT16);
196  // add integrity hash size
197  *value += sizeof(UINT16) +
198       CryptHashGetDigestSize(CONTEXT_INTEGRITY_HASH_ALG);
199  // Add fingerprint size, which is the same as sequence size
200  *value += sizeof(UINT64);
201  // Add OBJECT structure size
202  *value += sizeof(OBJECT);
203  #else
204  // the maximum size of a TPMS_CONTEXT that will be returned by
205  // TPM2_ContextSave for object context
206  *value = SIZE_OF_CTX_OVERHEAD + sizeof(OBJECT);
207  #endif
208  break;
209  case TPM_PT_MAX_SESSION_CONTEXT:
210  #if 0
211  // the maximum size of a TPMS_CONTEXT that will be returned by
212  // TPM2_ContextSave for object context
213  *value = 0;
214  // adding sequence, saved handle and hierarchy
215  *value += sizeof(UINT64) + sizeof(TPMI_DH_CONTEXT) +
216       sizeof(TPMI RH HIERARCHY);
217  // Add size field in TPM2B CONTEXT
218  *value += sizeof(UINT16);
219  // Add integrity hash size
220  *value += sizeof(UINT16) +
221       CryptHashGetDigestSize(CONTEXT_INTEGRITY_HASH_ALG);
222  // Add fingerprint size, which is the same as sequence size
223  *value += sizeof(UINT64);
224  // Add SESSION structure size
225  *value += sizeof(SESSION);
226  #else
227  // the maximum size of a TPMS_CONTEXT that will be returned by
228  // TPM2_ContextSave for object context
229  *value = SIZE_OF_CTX_OVERHEAD + sizeof(SESSION);
230  #endif
231  break;
232  case TPM_PT_PS_FAMILY_INDICATOR:
233  // platform specific values for the TPM_PT_PS parameters from
234  // the relevant platform-specific specification
In this reference implementation, all of these values are 0.

*value = PLATFORM_FAMILY;
break;

case TPM_PT_PS_LEVEL:
   // level of the platform-specific specification
   *value = PLATFORM_LEVEL;
break;

case TPM_PT_PS_REVISION:
   // specification Revision times 100 for the platform-specific
   // specification
   *value = PLATFORM_VERSION;
break;

case TPM_PT_PS_DAY_OF_YEAR:
   // platform-specific specification day of year using TCG calendar
   *value = PLATFORM_DAY_OF_YEAR;
break;

case TPM_PT_PS_YEAR:
   // platform-specific specification year using the CE
   *value = PLATFORM_YEAR;
break;

case TPM_PT_SPLIT_MAX:
   // number of split signing operations supported by the TPM
   *value = 0;
   #if ALG_ECC
      *value = sizeof(gr.commitArray) * 8;
   #endif
break;

case TPM_PT_TOTAL_COMMANDS:
   // total number of commands implemented in the TPM
   // Since the reference implementation does not have any
   // vendor-defined commands, this will be the same as the
   // number of library commands.
   {
      #if COMPRESSED_LISTS
         (*value) = COMMAND_COUNT;
      #else
         COMMAND_INDEX   commandIndex;
         *value = 0;

         // scan all implemented commands
         for(commandIndex = GetClosestCommandIndex(0);
             commandIndex != UNIMPLEMENTED_COMMAND_INDEX;
             commandIndex = GetNextCommandIndex(commandIndex))
            {
               (*value)++;
            // count of all implemented
            }
      #endif
   break;
}

case TPM_PT_LIBRARY_COMMANDS:
   // number of commands from the TPM library that are implemented
   {
      #if COMPRESSED_LISTS
         *value = LIBRARY_COMMAND_ARRAY_SIZE;
      #else
         COMMAND_INDEX   commandIndex;
         *value = 0;

         // scan all implemented commands
         for(commandIndex = GetClosestCommandIndex(0);
             commandIndex < LIBRARY_COMMAND_ARRAY_SIZE;
             commandIndex = GetNextCommandIndex(commandIndex))
            {
               (*value)++;
            }
      #endif
   }
Part 4: Supporting Routines

Trusted Platform Module Library

case TPM_PT_VENDOR_COMMANDS:
    // number of vendor commands that are implemented
    *value = VENDOR_COMMAND_ARRAY_SIZE;
    break;

case TPM_PT_NV_BUFFER_MAX:
    // Maximum data size in an NV write command
    *value = MAX_NV_BUFFER_SIZE;
    break;

case TPM_PT_MODES:
    #if FIPS_COMPLIANT
    *value = 1;
    #else
    *value = 0;
    #endif
    break;

case TPM_PT_MAX_CAP_BUFFER:
    *value = MAX_CAP_BUFFER;
    break;

    // Start of variable commands

case TPM_PT_PERMANENT:
    // TPMA_PERMANENT
    {
    union {
        TPMA_PERMANENT         attr;
        UINT32              u32;
    } flags = {{0}};
    if(gp.ownerAuth.t.size != 0)
    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, ownerAuthSet);
    if(gp.endorsementAuth.t.size != 0)
    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, endorsementAuthSet);
    if(gp.lockoutAuth.t.size != 0)
    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, lockoutAuthSet);
    if(gp.disableClear)
    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, disableClear);
    if(gp.failedTries >= gp.maxTries)
    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, inLockout);
    // In this implementation, EPS is always generated by TPM
    SET_ATTRIBUTE(flags.attr, TPMA_PERMANENT, tpmGeneratedEPS);
    // Note: For a LSB0 machine, the bits in a bit field are in the correct
    // order even if the machine is MSB0. For a MSB0 machine, a TPMA will
    // be an integer manipulated by masking (USE_BIT_FIELD_STRUCTURES will
    // be NO) so the bits are manipulated correctly.
    *value = flags.u32;
    break;
    }

case TPM_PT_STARTUP_CLEAR:
    // TPMA_STARTUP_CLEAR
    {
    union {
        TPMA_STARTUP_CLEAR attr;
        UINT32        u32;
    } flags = {{0}};
    //
    if(gp.shEnable)
    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, phEnable);
    if(gc.shEnable)
    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, shEnable);
    if(gc.ehEnable)
    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, ehEnable);
    if(gc.phEnableNV)
    SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, phEnableNV);
    if(g_prevOrderlyState != SU_NONE_VALUE)
368     SET_ATTRIBUTE(flags.attr, TPMA_STARTUP_CLEAR, orderly);
369
370     // Note: For a LSB0 machine, the bits in a bit field are in the correct
371     // order even if the machine is MSB0. For a MSb0 machine, a TPMA will
372     // be an integer manipulated by masking (USE_BIT_FIELD_STRUCTURES will
373     // be NO) so the bits are manipulate correctly.
374     *value = flags.u32;
375     break;
376 }
377
378 case TPM_PT_HR_NV_INDEX:
379     // number of NV indexes currently defined
380     *value = NvCapGetIndexNumber();
381     break;
382 case TPM_PT_HR_LOADED:
383     // number of authorization sessions currently loaded into TPM
384     // RAM
385     *value = SessionCapGetLoadedNumber();
386     break;
387 case TPM_PT_HR_LOADED_AVAIL:
388     // number of additional authorization sessions, of any type,
389     // that could be loaded into TPM RAM
390     *value = SessionCapGetLoadedAvail();
391     break;
392 case TPM_PT_HR_ACTIVE:
393     // number of active authorization sessions currently being
394     // tracked by the TPM
395     *value = SessionCapGetActiveNumber();
396     break;
397 case TPM_PT_HR_ACTIVE_AVAIL:
398     // number of additional authorization sessions, of any type,
399     // that could be created
400     *value = SessionCapGetActiveAvail();
401     break;
402 case TPM_PT_HR_TRANIENT_AVAIL:
403     // estimate of the number of additional transient objects that
404     // could be loaded into TPM RAM
405     *value = ObjectCapGetTransientAvail();
406     break;
407 case TPM_PT_HR_PERSISTENT:
408     // number of persistent objects currently loaded into TPM
409     // NV memory
410     *value = NvCapGetPersistentNumber();
411     break;
412 case TPM_PT_HR_PERSISTENT_AVAIL:
413     // number of additional persistent objects that could be loaded
414     // into NV memory
415     *value = NvCapGetPersistentAvail();
416     break;
417 case TPM_PT_NV_COUNTERS:
418     // number of defined NV indexes that have NV TPMA_NV_COUNTER
419     // attribute SET
420     *value = NvCapGetCounterNumber();
421     break;
422 case TPM_PT_NV_COUNTERS_AVAIL:
423     // number of additional NV indexes that can be defined with their
424     // TPMA_NV_COUNTER attribute SET
425     *value = NvCapGetCounterAvail();
426     break;
427 case TPM_PT_ALGORITHM_SET:
428     // region code for the TPM
429     *value = gp.algorithmSet;
430     break;
431 #if ALG_ECC
432     // number of loaded ECC curves
433     *value = ECC_CURVE_COUNT;
434 #endif
#else // ALG_ECC
  *value = 0;
#endif // ALG_ECC

  case TPM_PT_LOCKOUT_COUNTER:
    // current value of the lockout counter
    *value = gp.failedTries;
    break;
  case TPM_PT_MAX_AUTH_FAIL:
    // number of authorization failures before DA lockout is invoked
    *value = gp.maxTries;
    break;
  case TPM_PT_LOCKOUT_INTERVAL:
    // number of seconds before the value reported by
    // TPM_PT_LOCKOUT_COUNTER is decremented
    *value = gp.recoveryTime;
    break;
  case TPM_PT_LOCKOUT_RECOVERY:
    // number of seconds after a lockoutAuth failure before use of
    // lockoutAuth may be attempted again
    *value = gp.lockoutRecovery;
    break;
  case TPM_PT_NV_WRITE_RECOVERY:
    // number of milliseconds before the TPM will accept another command
    // that will modify NV.
    // This should make a call to the platform code that is doing rate
    // limiting of NV. Rate limiting is not implemented in the reference
    // code so no call is made.
    *value = 0;
    break;
  case TPM_PT_AUDIT_COUNTER_0:
    // high-order 32 bits of the command audit counter
    *value = (UINT32)(gp.auditCounter >> 32);
    break;
  case TPM_PT_AUDIT_COUNTER_1:
    // low-order 32 bits of the command audit counter
    *value = (UINT32)(gp.auditCounter);
    break;
  default:
    // property is not defined
    return FALSE;
    break;
  }
return TRUE;
}

9.14.3.2 TPMCapGetProperties()

This function is used to get the TPM_PT values. The search of properties will start at property and continue until propertyList has as many values as will fit, or the last property has been reported, or the list has as many values as requested in count.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>more properties are available</td>
</tr>
<tr>
<td>NO</td>
<td>no more properties to be reported</td>
</tr>
</tbody>
</table>

TPMI_YES_NO

TPMCapGetProperties(
  TPM_PT                       property, // IN: the starting TPM property
  UINT32                       count,  // IN: maximum number of returned
                                 // properties
  TPML_TAGGED_TPM_PROPERTY    *propertyList // OUT: property list
)
{ TPML_YES_NO more = NO;
 UINT32 i;
 UINT32 nextGroup;

 // initialize output property list
 propertyList->count = 0;

 // maximum count of properties we may return is MAX_PCR_PROPERTIES
 if(count > MAX_TPM_PROPERTIES) count = MAX_TPM_PROPERTIES;

 // if property is less than PT_FIXED, start from PT_FIXED
 if(property < PT_FIXED)
  property = PT_FIXED;

 // There is only the fixed and variable groups with the variable group coming
 // last
 if(property >= (PT_VAR + PT_GROUP))
  return more;

 // Don't read past the end of the selected group
 nextGroup = ((property / PT_GROUP) * PT_GROUP) + PT_GROUP;

 // Scan through the TPM properties of the requested group.
 for(i = property; i < nextGroup; i++)
  {
   UINT32 value;
   // if we have hit the end of the group, quit
   if(i != property && ((i % PT_GROUP) == 0))
    break;
   if(TPMPropertyIsDefined((TPM_PT)i, &value))
    {
     if(propertyList->count < count)
      {
       // If the list is not full, add this property
       propertyList->tpmProperty[propertyList->count].property =
          (TPM_PT)i;
       propertyList->tpmProperty[propertyList->count].value = value;
       propertyList->count++;
      }
     else
      {
       // If the return list is full but there are more properties
       // available, set the indication and exit the loop.
       more = YES;
       break;
      }
    }
   }
 return more;
9.15 Response.c

9.15.1 Description

This file contains the common code for building a response header, including setting the size of the structure. command may be NULL if result is not TPM_RC_SUCCESS.

9.15.2 Includes and Defines

```
#include "Tpm.h"
```

9.15.3 BuildResponseHeader()

Adds the response header to the response. It will update command->parameterSize to indicate the total size of the response.

```
void BuildResponseHeader(
    COMMAND *command, // IN: main control structure
    BYTE *buffer, // OUT: the output buffer
    TPM_RC result // IN: the response code
) {
    TPM_ST tag;
    UINT32 size;

    if(result != TPM_RC_SUCCESS)
        {
            tag = TPM_ST_NO_SESSIONS;
            size = 10;
        }
    else
    {
        tag = command->tag;
        // Compute the overall size of the response
        size = STD_RESPONSE_HEADER + command->handleNum * sizeof(TPM_HANDLE);
        size += command->parameterSize;
        size += (command->tag == TPM_ST_SESSIONS) ? command->authSize + sizeof(UINT32) : 0;
    }
    TPM_ST.Marshal(&tag, &buffer, NULL);
    UINT32.Marshal(&size, &buffer, NULL);
    TPM_RC.Marshal(&result, &buffer, NULL);
    if(result == TPM_RC_SUCCESS)
    {
        if(command->handleNum > 0)
            TPM_HANDLE.Marshal(&command->handles[0], &buffer, NULL);
        if(tag == TPM_ST_SESSIONS)
            UINT32.Marshal((UINT32 *)&command->parameterSize, &buffer, NULL);
    }
    command->parameterSize = size;
}
```
9.16  ResponseCodeProcessing.c

9.16.1  Description

This file contains the miscellaneous functions for processing response codes.

NOTE: Currently, there is only one.

9.16.2  Includes and Defines

```c
#include "Tpm.h"
```

9.16.3  RcSafeAddToResult()

Adds a modifier to a response code as long as the response code allows a modifier and no modifier has already been added.

```c
TPM_RC
RcSafeAddToResult(
    TPM_RC responseCode,
    TPM_RC modifier
)
{
    if((responseCode & RC_FMT1) && !(responseCode & 0xf40))
        return responseCode + modifier;
    else
        return responseCode;
}
```
9.17 TpmFail.c

9.17.1 Includes, Defines, and Types

```c
#define TPM_FAIL_C
#include "Tpm.h"
#include <assert.h>
```

On MS C compiler, can save the alignment state and set the alignment to 1 for the duration of the TpmTypes.h include. This will avoid a lot of alignment warnings from the compiler for the unaligned structures. The alignment of the structures is not important as this function does not use any of the structures in TpmTypes.h and only include it for the #defines of the capabilities, properties, and command code values.

```c
#include "TpmTypes.h"
```

9.17.2 Typedefs

These defines are used primarily for sizing of the local response buffer.

```c
typedef struct {
    TPM_ST          tag;
    UINT32          size;
    TPM_RC          code;
} HEADER;

typedef struct {
    BYTE            tag[sizeof(TPM_ST)];
    BYTE            size[sizeof(UINT32)];
    BYTE            code[sizeof(TPM_RC)];
} PACKED_HEADER;

typedef struct {
    BYTE             size[sizeof(UINT16)];
    struct {
        BYTE          function[sizeof(UINT32)];
        BYTE          line[sizeof(UINT32)];
        BYTE          code[sizeof(UINT32)];
    } values;
    BYTE             returnCode[sizeof(TPM_RC)];
} GET_TEST_RESULT_PARAMETERS;

typedef struct {
    BYTE             header[sizeof(PACKED_HEADER)];
    BYTE             getTestResult[sizeof(GET_TEST_RESULT_PARAMETERS)];
} TEST_RESPONSE;

typedef struct {
    BYTE             header[sizeof(PACKED_HEADER)];
    BYTE             getCap[sizeof(GET_CAPABILITY_PARAMETERS)];
} CAPABILITY_RESPONSE;
```

```c
typedef union {
```

```
```
BYTE test[sizeof(TEST_RESPONSE)];
BYTE cap[sizeof(CAPABILITY_RESPONSE)];
RESPONSES;

Buffer to hold the responses. This may be a little larger than required due to padding that a compiler might add.

NOTE: This is not in Global.c because of the specialized data definitions above. Since the data contained in this structure is not relevant outside of the execution of a single command (when the TPM is in failure mode. There is no compelling reason to move all the typedefs to Global.h and this structure to Global.c.

ifndef __IGNORE_STATE__ // Don't define this value
static BYTE response[sizeof(RESPONSES)];
endif

9.17.3 Local Functions

9.17.3.1 MarshalUint16()

Function to marshal a 16 bit value to the output buffer.

static INT32 MarshalUint16(
    UINT16 integer,
    BYTE **buffer
){
    return UINT16_Marshal(&integer, buffer, NULL);
}

9.17.3.2 MarshalUint32()

Function to marshal a 32 bit value to the output buffer.

static INT32 MarshalUint32(
    UINT32 integer,
    BYTE **buffer
){
    return UINT32_Marshal(&integer, buffer, NULL);
}

9.17.3.3 UnmarshalHeader()

function to unmarshal the 10-byte command header.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure</td>
</tr>
</tbody>
</table>

static BOOL UnmarshalHeader(
    HEADER *header,
    BYTE **buffer,
    INT32 *size
){

9.17.4 Public Functions

9.17.4.1 SetForceFailureMode()

This function is called by the simulator to enable failure mode testing.

```c
#if SIMULATION
LIB_EXPORT void
SetForceFailureMode(
    void
)
{
    g_forceFailureMode = TRUE;
    return;
}
#endif
```

9.17.4.2 TpmFail()

This function is called by TPM.lib when a failure occurs. It will set up the failure values to be returned on TPM2_GetTestResult().

```c
NORETURN void
TpmFail(
    #if FAIL_TRACE
        const char *function,
        int line,
    #endif
    int code
)
{
    // Save the values that indicate where the error occurred.
    // On a 64-bit machine, this may truncate the address of the string
    // of the function name where the error occurred.
    #if FAIL_TRACE
        s_failFunction = (UINT32)function;
        s_failLine = line;
    #else
        s_failFunction = (UINT32)NULL;
        s_failLine = 0;
    #endif
    s_failCode = code;

    // We are in failure mode
    g_inFailureMode = TRUE;

    // if asserts are enabled, then do an assert unless the failure mode code
    // is being tested.
    #if SIMULATION
        #ifndef NDEBUG
            assert(g_forceFailureMode);
        #endif
    ```
9.17.4.3 TpmFailureMode

This function is called by the interface code when the platform is in failure mode.

```c
void TpmFailureMode(
    unsigned int inRequestSize,  // IN: command buffer size
    unsigned char *inRequest,   // IN: command buffer
    unsigned int *outResponseSize, // OUT: response buffer size
    unsigned char **outResponse  // OUT: response buffer
) {
    BYTE *buffer;
    UINT32 marshalSize;
    UINT32 capability;
    HEADER header;  // unmarshaled command header
    UINT32 pt;     // unmarshaled property type
    UINT32 count;  // unmarshaled property count

    // If there is no command buffer, then just return TPM_RC_FAILURE
    if (inRequestSize == 0 || inRequest == NULL)
        goto FailureModeReturn;

    // If the header is not correct for TPM2_GetCapability() or
    // TPM2_GetTestResult() then just return the in failure mode response;
    buffer = inRequest:
    if (!UnmarshalHeader(&header, &inRequest, (INT32 *)&inRequestSize))
        goto FailureModeReturn;
    if (header.tag != TPM_ST_NO_SESSIONS || header.size < 10)
        goto FailureModeReturn;

    switch (header.code) {
        case TPM_CC_GetTestResult:
            // make sure that the command size is correct
            if (header.size != 10)
                goto FailureModeReturn;
            buffer = &response[10];
            marshalSize = MarshalUint16(3 * sizeof(UINT32), &buffer);
            marshalSize += MarshalUint32(s_failFunction, &buffer);
            marshalSize += MarshalUint32(s_failLine, &buffer);
            marshalSize += MarshalUint32(s_failCode, &buffer);
            if (s_failCode == FATAL_ERROR_NV_UNRECOVERABLE)
                marshalSize += MarshalUint32(TPM_RC_NV_UNINITIALIZED, &buffer);
            else
                marshalSize += MarshalUint32(TPM_RC_FAILURE, &buffer);
            break;

        case TPM_CC_GetCapability:
            // make sure that the size of the command is exactly the size
            // returned for the capability, property, and count
            if (header.size != (10 + (3 * sizeof(UINT32))))
                goto FailureModeReturn;
            // also verify that this is requesting TPM properties
            if (TPM_RC_SUCCESS != UINT32_Unmarshal(&capability, &inRequest, (INT32 *)&inRequestSize))
                goto FailureModeReturn;
            if (capability != TPM_CAP_TPM_PROPERTIES)
                goto FailureModeReturn;
            break;
    }
}
```
183 (INT32 *)&inRequestSize)
184 || TPM_RC_SUCCESS != UINT32_Unmarshal(&count, &inRequest,
185 (INT32 *)&inRequestSize))
186 goto FailureModeReturn;
187 // If in failure mode because of an unrecoverable read error, and the
188 // property is 0 and the count is 0, then this is an indication to
189 // re-manufacture the TPM. Do the re-manufacture but stay in failure
190 // mode until the TPM is reset.
191 // Note: this behavior is not required by the specification and it is
192 // OK to leave the TPM permanently bricked due to an unrecoverable NV
193 // error.
194 if(count == 0 && pt == 0 && s_failCode == FATAL_ERROR_NV_UNRECOVERABLE)
195 {
196     g_manufactured = FALSE;
197     TPM_Manufacture(0);
198 }
199 if(count > 0)
200     count = 1;
201 else if(pt > TPM_PT_FIRMWARE_VERSION_2)
202     count = 0;
203 if(pt < TPM_PT_MANUFACTURER)
204     pt = TPM_PT_MANUFACTURER;
205 // set up for return
206 buffer = &response[10];
207 // if the request was for a PT less than the last one
208 // then we indicate more, otherwise, not.
209 if(pt < TPM_PT_FIRMWARE_VERSION_2)
210     *buffer++ = YES;
211 else
212     *buffer++ = NO;
213 marshalSize = 1;
214 // indicate the capability type
215 marshalSize += MarshalUint32(capability, &buffer);
216 // indicate the number of values that are being returned (0 or 1)
217 marshalSize += MarshalUint32(count, &buffer);
218 // indicate the property
219 marshalSize += MarshalUint32(pt, &buffer);
220 if(count > 0)
221     switch(pt)
222 {
223     case TPM_PT_MANUFACTURER:
224         // the vendor ID unique to each TPM manufacturer
225         #ifdef MANUFACTURER
226             pt = *(UINT32*)MANUFACTURER;
227         #else
228             pt = 0;
229         #endif
230         break;
231     case TPM_PT_VENDOR_STRING_1:
232         // the first four characters of the vendor ID string
233         #ifdef VENDOR_STRING_1
234             pt = *(UINT32*)VENDOR_STRING_1;
235         #else
236             pt = 0;
237         #endif
238         break;
239     case TPM_PT_VENDOR_STRING_2:
240         // the second four characters of the vendor ID string
241         #ifdef VENDOR_STRING_2
242             pt = *(UINT32*)VENDOR_STRING_2;
243         #else
244             pt = 0;
245         #endif
246         break;
case TPM_PT_VENDOR_STRING_3:
    // the third four characters of the vendor ID string
    #ifdef VENDOR_STRING_3
    pt = *(UINT32*)VENDOR_STRING_3;
    #else
    pt = 0;
    #endif
    break;
#endif
#endif
#endif
case TPM_PT_VENDOR_STRING_4:
    // the fourth four characters of the vendor ID string
    #ifdef VENDOR_STRING_4
    pt = *(UINT32*)VENDOR_STRING_4;
    #else
    pt = 0;
    #endif
    break;
#endif
#endif
#endif
#endif
case TPM_PT_VENDOR_TPM_TYPE:
    // vendor-defined value indicating the TPM model
    // We just make up a number here
    pt = 1;
    break;
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif
default: // TPM_PT_VENDOR_TPM_TYPE:
    // the more significant 32-bits of a vendor-specific value
    // indicating the version of the firmware
    #ifdef Firmware_V1
    pt = Firmware_V1;
    #else
    pt = 0;
    #endif
    break;
endif
default: // default for switch (cc)
    goto FailureModeReturn;
}
// Now do the header
buffer = response;
marshalSize = marshalSize + 10; // Add the header size to the
    // stuff already marshaled
MarshalUint16(TPM_ST_NO_SESSIONS, &buffer); // structure tag
MarshalUint32(marshalSize, &buffer); // responseSize
MarshalUint32(TPM_RC_SUCCESS, &buffer); // response code
*outResponseSize = marshalSize;
*outResponse = (unsigned char *)&response;
return;
FailureModeReturn:
    buffer = response;
    marshalSize = MarshalUint16(TPM_ST_NO_SESSIONS, &buffer);
    marshalSize += MarshalUint32(10, &buffer);
    marshalSize += MarshalUint32(TPM_RC_FAILURE, &buffer);
    *outResponseSize = marshalSize;
    *outResponse = (unsigned char *)response;
    return;
9.17.4.4 UnmarshalFail()

This is a stub that is used to catch an attempt to unmarshal an entry that is not defined. Don't ever expect this to be called but...

```c
void UnmarshalFail(  
    void *type,  
    BYTE **buffer,  
    INT32 *size  
)  
{
    NOT_REFERENCED(type);  
    NOT_REFERENCED(buffer);  
    NOT_REFERENCED(size);  
    FAIL(FATAL_ERROR_INTERNAL);  
}
```
10 Cryptographic Functions

10.1 Headers

10.1.1 BnValues.h

10.1.1.1 Introduction

This file contains the definitions needed for defining the internal BIGNUM structure. A BIGNUM is a pointer to a structure. The structure has three fields. The last field is an array (d) of crypt_uword_t. Each word is in machine format (big- or little-endian) with the words in ascending significance (i.e. words in little-endian order). This is the order that seems to be used in every big number library in the worlds, so...

The first field in the structure (allocated) is the number of words in d. This is the upper limit on the size of the number that can be held in the structure. This differs from libraries like OpenSSL() as this is not intended to deal with numbers of arbitrary size; just numbers that are needed to deal with the algorithms that are defined in the TPM implementation.

The second field in the structure (size) is the number of significant words in n. When this number is zero, the number is zero. The word at used-1 should never be zero. All words between d[size] and d[allocated-1] should be zero.

10.1.1.2 Defines

```c
#ifndef _BN_NUMBERS_H
#define _BN_NUMBERS_H
#if RADIX_BITS == 64
#define RADIX_LOG2 6
#else
#define RADIX_LOG2 5
#endif
#else "Unsupported radix"
#endif
#define RADIX_MOD(x) ((x) & ((1 << RADIX_LOG2) - 1))
#define RADIX_DIV(x) ((x) >> RADIX_LOG2)
#define RADIX_MASK ((((crypt_uword_t)1) << RADIX_LOG2) - 1)
#define BITS_TO_CRYPT_WORDS(bits) (RADIX_DIV((bits) + (RADIX_BITS - 1)))
#define BYTES_TO_CRYPT_WORDS(bytes) BITS_TO_CRYPT_WORDS(bytes * 8)
#define SIZE_IN_CRYPT_WORDS(thing) BYTES_TO_CRYPT_WORDS(sizeof(thing))
#if RADIX_BITS == 64
#define SWAP_CRYPT_WORD(x) REVERSE_ENDIAN_64(x)
typedef uint64_t crypt_uword_t;
typedef int64_t crypt_word_t;
#else
#define SWAP_CRYPT_WORD(x) REVERSE_ENDIAN_32((x))
typedef uint32_t crypt_uword_t;
typedef int32_t crypt_word_t;
#endif
#define MAX_CRYPT_UWORD (~((crypt_uword_t)0))
#define MAX_CRYPT_WORD  ((crypt_word_t)(MAX_CRYPT_UWORD >> 1))
#define MIN_CRYPT_WORD  (~MAX_CRYPT_WORD)
#define LARGEST_NUMBER (MAX((ALG_RSA * MAX_RSA_KEY_BYTES), MAX((ALG_ECC * MAX_ECC_KEY_BYTES), MAX_DIGEST_SIZE)))
#define LARGEST_NUMBER_BITS (LARGEST_NUMBER * 8)
#define MAX_ECC_PARAMETER_BYTES (MAX_ECC_KEY_BYTES * ALG_ECC)
```
These are the basic big number formats. This is convertible to the library-specific format without too much difficulty. For the math performed using these numbers, the value is always positive.

```
#define BN_STRUCT_DEF(count) struct {
    crypt_uword_t allocated;
    crypt_uword_t size;
    crypt_uword_t d[count];
}

typedef BN_STRUCT_DEF(1) bignum_t;
#ifndef bigNum
typedef bignum_t *bigNum;
typedef const bignum_t *bigConst;
#endif
extern const bignum_t BnConstZero;
```

The Functions to access the properties of a big number. Get number of allocated words

```
#define BnGetAllocated(x)   ((unsigned)((x)->allocated)

#define BnGetSize(x)        ((x)->size)

#define BnGetArray(x)       ((crypt_uword_t *)&((x)->d[0]))

#define BnGetWord(x, i)     (crypt_uword_t)((x)->d[i])
```

Some things that are done often. Test to see if a bignum_t is equal to zero

```
#define BnEqualZero(bn)   (BnGetSize(bn) == 0)

#define BnEqualWord(bn, word)
    ((BnGetSize(bn) == 1) && (BnGetWord(bn, 0) == (crypt_uword_t)word))
```

Determine if a BIGNUM is even. A zero is even. Although the indication that a number is zero is that it's size is zero, all words of the number are 0 so this test works on zero.

```
#define BnIsEven(n)     ((BnGetWord(n, 0) & 1) == 0)
```

Test to see if a bignum_t is equal to a word type

```
#define BnEqualWord(bn, word)
    ((BnGetSize(bn) == 1) && (BnGetWord(bn, 0) == (crypt_uword_t)word))
```

Determine if a BIGNUM is even. A zero is even. Although the indication that a number is zero is that it's size is zero, all words of the number are 0 so this test works on zero.

```
#define BnIsEven(n)     ((BnGetWord(n, 0) & 1) == 0)
```

The macros below are used to define BIGNUM values of the required size. The values are allocated on the stack so they can be treated like simple local values. This will call the initialization function for a defined bignum_t. This sets the allocated and used fields and clears the words of n.

```
#define BN_INIT(name)
    (bigNum)BnInit((bigNum)&(name),
                   BYTES_TO_CRYPT_WORDS(sizeof(name.d)))
```

In some cases, a function will need the address of the structure associated with a variable. The structure for a BIGNUM variable of name is name_. Generally, when the structure is created, it is initialized and a parameter is created with a pointer to the structure. The pointer has the name and the structure it points to is name_

```
#define BN_ADDRESS(name) (bigNum)&name##_
#define BN_STRUCT_ALLOCATION(bits) (BITS_TO_CRYPT_WORDS(bits) + 1)
```
Create a structure of the correct size.

```c
#define BN_STRUCT(bits) \
   BN_STRUCT_DEF(BN_STRUCT_ALLOCATION(bits))
```

Define a BIGNUM type with a specific allocation

```c
#define BN_TYPE(name, bits) \
   typedef BN_STRUCT(bits) bn_##name##_t
```

This creates a local BIGNUM variable of a specific size and initializes it from a TPM2B input parameter.

```c
#define BN_INITIALIZED(name, bits, initializer) \
   BN_STRUCT<bits> name##_; \
   bigNum name = BnFrom2B(BN_INIT(name##_), (const TPM2B *)initializer)
```

Create a local variable that can hold a number with bits

```c
#define BN_VAR(name, bits) \
   BN_STRUCT<bits> _##name; \
   bigNum name = BN_INIT(_##name)
```

Create a type that can hold the largest number defined by the implementation.

```c
#define BN_MAX(name)   BN_VAR(name, LARGEST_NUMBER_BITS) \
#define BN_MAX_INITIALIZED(name, initializer) \
   BN_INITIALIZED(name, LARGEST_NUMBER_BITS, initializer)
```

A word size value is useful

```c
#define BN_WORD(name)      BN_VAR(name, RADIX_BITS)
```

This is used to created a word-size BIGNUM and initialize it with an input parameter to a function.

```c
#define BN_WORD_INITIALIZED(name, initial) \
   BN_INITIALIZED(name, RADIX_BITS, initial)
```

ECC-Specific Values This is the format for a point. It is always in affine format. The Z value is carried as part of the point, primarily to simplify the interface to the support library. Rather than have the interface layer have to create space for the point each time it is used... The x, y, and z values are pointers to bigNum values and not in-line versions of the numbers. This is a relic of the days when there was no standard TPM format for the numbers

```c
typedef struct _bn_point_t \
{ \
   bigNum x; \
   bigNum y; \
   bigNum z; \
} bn_point_t; \
```

```c
typedef bn_point_t *bigPoint; \
```

```c
typedef const bn_point_t *pointConst; 
```

```c
typedef struct constant_point_t \
{ \
   bigConst x; \
   bigConst y; \
   bigConst z; \
} constant_point_t; 
```

```c
#define ECC_BITS (MAX_ECC_KEY_BYTES * 8)
```
95 BN_TYPE(ecc, ECC_BITS);
96 #define ECC_NUM(name) BN_VAR(name, ECC_BITS)
97 #define ECC_INITIALIZED(name, initializer) \
98 BN_INITIALIZED(name, ECC_BITS, initializer)
99 #define POINT_INSTANCE(name, bits) \
100 BN_STRUCT (bits) name##_x = \
101 {BITS_TOCRYPT_WORDS (bits), 0, {0}};
102 BN_STRUCT (bits) name##_y = \
103 {BITS_TOCRYPT_WORDS (bits), 0, {0}};
104 BN_STRUCT (bits) name##_z = \
105 {BITS_TOCRYPT_WORDS (bits), 0, {0}};
106 bn_point_t name##_t = \
107 #define POINT_INITIALIZER(name) \
108 BnInitializePoint(&name##_t, (bigNum)&name##_x, \
109 (bigNum)&name##_y, (bigNum)&name##_z)
110 #define POINT_INITIALIZED(name, initValue) \
111 POINT_INSTANCE(name, MAX_ECC_KEY_BITS);
112 bigPoint name = BnPointFrom2B( \
113 POINT_INITIALIZED(name), \
114 initValue)
115 #define POINT_VAR(name, bits) \
116 POINT_INSTANCE (name, bits); \
117 bigPoint name = POINT_INITIALIZER(name)
118 #define POINT(name) POINT_VAR(name, MAX_ECC_KEY_BITS)

Structure for the curve parameters. This is an analog to the TPMS_ALGORITHM_DETAIL_ECC

119 typedef struct
120 { 
121 bigConst prime; // a prime number
122 bigConst order; // the order of the curve
123 bigConst h; // cofactor
124 bigConst a; // linear coefficient
125 bigConst b; // constant term
126 constant_point_t base; // base point
127 } ECC_CURVE_DATA;

Access macros for the ECC_CURVE structure. The parameter C is a pointer to an ECC_CURVE_DATA structure. In some libraries, the curve structure contains a pointer to an ECC_CURVE_DATA structure as well as some other bits. For those cases, the AccessCurveData() macro is used in the code to first get the pointer to the ECC_CURVE_DATA for access. In some cases, the macro does noting.

128 #define CurveGetPrime(C) ((C)->prime)
129 #define CurveGetOrder(C) ((C)->order)
130 #define CurveGetCofactor(C) ((C)->h)
131 #define CurveGet_a(C) ((C)->a)
132 #define CurveGet_b(C) ((C)->b)
133 #define CurveGetG(C) ((pointConst)&((C)->base))
134 #define CurveGetGx(C) (C)->base.x)
135 #define CurveGetGy(C) (C)->base.y)

Convert bytes in initializers according to the endianess of the system. This is used for CryptEccData.c.

136 #define BIG_ENDIAN_BYTES_TO_UINT32(a, b, c, d) \
137 ( ((UINT32)(a) << 24) \
138 + ((UINT32)(b) << 16) \
139 + ((UINT32)(c) << 8) \
140 + ((UINT32)(d)) \
141 )
142 #define BIG_ENDIAN_BYTES_TO_UINT64(a, b, c, d, e, f, g, h) \
143 ( ((UINT64)(a) << 56) \
144 + ((UINT64)(b) << 48) \
145 + ((UINT64)(c) << 40) \
146 + ((UINT64)(d) << 32)
Add implementation dependent definitions for other ECC Values and for linkages. MATH_LIB_H is defined in LibSupport.h

```
147   +   ((UINT64)(e)  << 24) \n148   +   ((UINT64)(f)  << 16) \n149   +   ((UINT64)(g)  << 8) \n150   +   ((UINT64)(h)) \n151  )

#include MATHLIB_H
152 #ifndef RADIX_BYTES
153 #   if RADIX_BITS == 32
154 #       define RADIX_BYTES 4
155 #   elif RADIX_BITS == 64
156 #       define RADIX_BYTES 8
157 #   else
158 #       error "RADIX_BITS must either be 32 or 64"
159 #   endif
160 #endif
161 #endif // _BN_NUMBERS_H
```
10.1.2 CryptEcc.h

10.1.2.1 Introduction

This file contains structure definitions used for ECC. The structures in this file are only used internally. The ECC-related structures that cross the TPM interface are defined in TpmTypes.h

```
#ifndef _CRYPT_ECC_H
#define _CRYPT_ECC_H

10.1.2.2 Structures

This is used to define the macro that may or may not be in the data set for the curve (BnEccData.c). If there is a mismatch, the compiler will warn that there is to much/not enough initialization data in the curve. The macro is used because not all versions of the CryptEccData.c need the curve name.

```
10.1.3 CryptHash.h

10.1.3.1 Introduction

This header contains the hash structure definitions used in the TPM code to define the amount of space to be reserved for the hash state. This allows the TPM code to not have to import all of the symbols used by the hash computations. This lets the build environment of the TPM code not to have include the header files associated with the CryptoEngine() code.

```c
#ifndef _CRYPT_HASH_H
#define _CRYPT_HASH_H

10.1.3.2 Hash-related Structures

typedef struct {
    const TPM_ALG_ID     alg;
    const UINT16         digestSize;
    const UINT16         blockSize;
    const UINT16         derSize;
    const BYTE           der[20];
} HASH_INFO;

union SMAC_STATES;

These definitions add the high-level methods for processing state that may be an SMAC

typedef void(* SMAC_DATA_METHOD)(
    union SMAC_STATES *state,
    UINT32 size,
    const BYTE *buffer
);

typedef UINT16(* SMAC_END_METHOD)(
    union SMAC_STATES *state,
    UINT32 size,
    BYTE *buffer
);

typedef struct sequenceMethods {
    SMAC_DATA_METHOD          data;
    SMAC_END_METHOD           end;
} SMAC_METHODS;

#define SMAC_IMPLEMENTED (defined TPM_CC_MAC || defined TPM_CC_MAC_Start)

These definitions are here because the SMAC state is in the union of hash states.

typedef struct tpmCmacState {
    TPM_ALG_ID              symAlg;
    UINT16                  keySizeBits;
    INT16                   bcount; // current count of bytes accumulated in IV
    TPM2B_IV                iv; // IV buffer
    TPM2B_SYM_KEY           symKey;
} tpmCmacState_t;

typedef union SMAC_STATES {
    tpmCmacState_t          cmac;
    } SMAC_STATES;

typedef struct SMAC_STATE {
    SMAC_METHODS            smacMethods;
    SMAC_STATES             state;
} SMAC_STATE;
```
typedef union
{
  #if ALG_SHA1
    tpmHashStateSHA1_t   Sha1;
  #endif
  #if ALG_SHA256
    tpmHashStateSHA256_t Sha256;
  #endif
  #if ALG_SHA384
    tpmHashStateSHA384_t Sha384;
  #endif
  #if ALG_SHA512
    tpmHashStateSHA512_t Sha512;
  #endif

  // Additions for symmetric block cipher MAC
  #if SMAC_IMPLEMENTED
    SMAC_STATE          smac;
  #endif

  // to force structure alignment to be no worse than HASH_ALIGNMENT
  #if HASH_ALIGNMENT == 4
    uint32_t            align;
  #else
    uint64_t            align;
  #endif

} ANY_HASH_STATE;

typedef        ANY_HASH_STATE *PANY_HASH_STATE;
typedef const   ANY_HASH_STATE    *PCANY_HASH_STATE;
#define           ALIGNED_SIZE(x, b) ((((x) + (b) - 1) / (b)) * (b))

MAX_HASH_STATE_SIZE will change with each implementation. It is assumed that a hash state will not
be larger than twice the block size plus some overhead (in this case, 16 bytes). The overall size needs to
be as large as any of the hash contexts. The structure needs to start on an alignment boundary and be an
even multiple of the alignment

#define MAX_HASH_STATE_SIZE ((2 * MAX_HASH_BLOCK_SIZE) + 16)
#define MAX_HASH_STATE_SIZE_ALIGNED ALIGNED_SIZE(MAX_HASH_STATE_SIZE, HASH_ALIGNMENT)

This is an aligned byte array that will hold any of the hash contexts.

typedef    ANY_HASH_STATE ALIGNED_HASH_STATE;

The header associated with the hash library is expected to define the methods which include the calling
sequence. When not compiling CryptHash.c, the methods are not defined so we need placeholder
functions for the structures

#ifndef HASH_START_METHOD_DEF
#   define HASH_START_METHOD_DEF void (HASH_START_METHOD)(void)
#endif
#ifndef HASH_DATA_METHOD_DEF
#   define HASH_DATA_METHOD_DEF void (HASH_DATA_METHOD)(void)
#endif
#ifndef HASH_END_METHOD_DEF
#   define HASH_END_METHOD_DEF void (HASH_END_METHOD)(void)
#endif
#ifndef HASH_STATE_COPY_METHOD_DEF
#   define HASH_STATE_COPY_METHOD_DEF void (HASH_STATE_COPY_METHOD)(void)
#endif
#ifndef HASH_STATE_EXPORT_METHOD_DEF
#   define HASH_STATE_EXPORT_METHOD_DEF void (HASH_STATE_EXPORT_METHOD)(void)
#endif
#ifndef HASH_STATE_IMPORT_METHOD_DEF
#   define HASH_STATE_IMPORT_METHOD_DEF void (HASH_STATE_IMPORT_METHOD)(void)
#endif
# define HASH_STATE_IMPORT_METHOD_DEF  void (HASH_STATE_IMPORT_METHOD)(void)
#endif

Define the prototypical function call for each of the methods. This defines the order in which the parameters are passed to the underlying function.

typedef HASH_START_METHOD_DEF;
typedef HASH_DATA_METHOD_DEF;
typedef HASH_END_METHOD_DEF;
typedef HASH_STATE_COPY_METHOD_DEF;
typedef HASH_STATE_EXPORT_METHOD_DEF;
typedef HASH_STATE_IMPORT_METHOD_DEF;
typedef struct _HASH_METHODS
{
    HASH_START_METHOD           *start;
    HASH_DATA_METHOD            *data;
    HASH_END_METHOD             *end;
    HASH_STATE_COPY_METHOD      *copy;   // Copy a hash block
    HASH_STATE_EXPORT_METHOD    *copyOut; // Copy a hash block from a hash context
    HASH_STATE_IMPORT_METHOD    *copyIn;  // Copy a hash block to a proper hash context
}

) HASH_METHODS, *PHASH_METHODS;
#if ALG_SHA1
    TPM2B_TYPE(SHA1_DIGEST, SHA1_DIGEST_SIZE);
#endif
#if ALG_SHA256
    TPM2B_TYPE(SHA256_DIGEST, SHA256_DIGEST_SIZE);
#endif
#if ALG_SHA384
    TPM2B_TYPE(SHA384_DIGEST, SHA384_DIGEST_SIZE);
#endif
#if ALG_SHA512
    TPM2B_TYPE(SHA512_DIGEST, SHA512_DIGEST_SIZE);
#endif
#if ALG_SM3_256
    TPM2B_TYPE(SM3_256_DIGEST, SM3_256_DIGEST_SIZE);
#endif
typedef const struct
{
    HASH_METHODS         method;
    uint16_t             blockSize;
    uint16_t             digestSize;
    uint16_t             contextSize;
    uint16_t             hashAlg;
} HASH_DEF, *PHASH_DEF;

Macro to fill in the HASH_DEF for an algorithm. For SHA1, the instance would be: HASH_DEF_TEMPLATE(Sha1, SHA1) This handles the difference in capitalization for the various pieces.

#define HASH_DEF_TEMPLATE(HASH) \n    HASH_DEF  HASH##_Def= { \n    (HASH_START_METHOD *)&tpmHashStart_##HASH, 
    (HASH_DATA_METHOD *)&tpmHashData_##HASH, 
    (HASH_END_METHOD *)&tpmHashEnd_##HASH, 
    (HASH_STATE_COPY_METHOD *)&tpmHashStateCopy_##HASH, 
    (HASH_STATE_EXPORT_METHOD *)&tpmHashStateExport_##HASH, 
    (HASH_STATE_IMPORT_METHOD *)&tpmHashStateImport_##HASH, 
    },
    BLOCK_SIZE, /*block size */
    DIGEST_SIZE, /*data size */
    sizeof(tpmHashState##HASH##_t),
    TPM_ALG_##HASH}
These definitions are for the types that can be in a hash state structure. These types are used in the cryptographic utilities. This is a define rather than an enum so that the size of this field can be explicit.

```c
typedef BYTE HASH_STATE_TYPE;
#define HASH_STATE_EMPTY ((HASH_STATE_TYPE) 0)
#define HASH_STATE_HASH  ((HASH_STATE_TYPE) 1)
#define HASH_STATE_HMAC ((HASH_STATE_TYPE) 2)
#if CC_MAC || CC_MAC_Start
#define HASH_STATE_SMAC ((HASH_STATE_TYPE) 3)
#endif
```

This is the structure that is used for passing a context into the hashing functions. It should be the same size as the function context used within the hashing functions. This is checked when the hash function is initialized. This version uses a new layout for the contexts and a different definition. The state buffer is an array of HASH_UNIT values so that a decent compiler will put the structure on a HASH_UNIT boundary. If the structure is not properly aligned, the code that manipulates the structure will copy to a properly aligned structure before it is used and copy the result back. This just makes things slower.

NOTE: This version of the state had the pointer to the update method in the state. This is to allow the SMAC functions to use the same structure without having to replicate the entire HASH_DEF structure.

```c
typedef struct _HASH_STATE
{
    HASH_STATE_TYPE          type;       // type of the context
    TPM_ALG_ID               hashAlg;
    PHASH_DEF                def;
    ANY_HASH_STATE state;
} HASH_STATE, *PHASH_STATE;
```

10.1.3.3 HMAC State Structures

An HMAC_STATE structure contains an opaque HMAC stack state. A caller would use this structure when performing incremental HMAC operations. This structure contains a hash state and an HMAC key and allows slightly better stack optimization than adding an HMAC key to each hash state.

```c
typedef struct hmacState
{
    HASH_STATE hashState;    // the hash state
    TPM2B_HASH_BLOCK hmacKey;  // the HMAC key
} HMAC_STATE, *PHMAC_STATE;
```

This is for the external hash state. This implementation assumes that the size of the exported hash state is no larger than the internal hash state. There is a run time check that makes sure that this is
10.1.4 CryptRand.h

10.1.4.1 Introduction

This file contains constant definition shared by CryptUtil() and the parts of the Crypto Engine.

```c
#ifndef _CRYPT_RAND_H
#define _CRYPT_RAND_H

10.1.4.2 DRBG Structures and Defines

Values and structures for the random number generator. These values are defined in this header file so that the size of the RNG state can be known to TPM.lib. This allows the allocation of some space in NV memory for the state to be stored on an orderly shutdown. The DRBG based on a symmetric block cipher is defined by three values,

a) the key size
b) the block size (the IV size)
c) the symmetric algorithm

```c
#define DRBG_KEY_SIZE_BITS MAX_AES_KEY_BITS
#define DRBG_IV_SIZE_BITS (MAX_AES_BLOCK_SIZE_BYTES * 8)
#define DRBG_ALGORITHM TPM_ALG_AES

typedef tpmKeyScheduleAES DRBG_KEY_SCHEDULE;
#define DRBG_ENCRIPT_SETUP(key, keySizeInBits, schedule) \ 
   TpmCryptSetEncryptKeyAES(key, keySizeInBits, schedule) \ 
   TpmCryptEncryptAES(SWIZZLE(keySchedule, in, out))

#elif ((DRBG_KEY_SIZE_BITS % RADIX_BITS) != 0) \ 
   || ((DRBG_IV_SIZE_BITS % RADIX_BITS) != 0)
#error "Key size and IV for DRBG must be even multiples of the radix"
#endif
#elif (DRBG_KEY_SIZE_BITS % DRBG_IV_SIZE_BITS) != 0
#error "Key size for DRBG must be even multiple of the cipher block size"
#endif

Derived values

```c
#define DRBG_MAX_REQUESTS_PER_RESEED (1 << 48)
#define DRBG_MAX_REQUEST_SIZE (1 << 32)
#define pDRBG_KEY(seed)   ((DRBG_KEY *)(&(BYTE *)(seed)[0]))
#define pDRBG_IV(seed)    ((DRBG_IV *)(&(BYTE *)(seed)[DRBG_KEY_SIZE_BYTES]))
#define DRBG_KEY_SIZE_WORDS (BITS_TO_CRYPT_WORDS(DRBG_KEY_SIZE_BITS))
#define DRBG_KEY_SIZE_BYTES (DRBG_KEY_SIZE_WORDS * RADIX_BYTES)
#define DRBG_IV_SIZE_WORDS (BITS_TO_CRYPT_WORDS(DRBG_IV_SIZE_BITS))
#define DRBG_IV_SIZE_BYTES (DRBG_IV_SIZE_WORDS * RADIX_BYTES)
#define DRBG_SEED_SIZE_WORDS (DRBG_KEY_SIZE_WORDS + DRBG_IV_SIZE_WORDS)
#define DRBG_SEED_SIZE_BYTES (DRBG_KEY_SIZE_BYTES + DRBG_IV_SIZE_BYTES)

typedef union
{
   BYTE bytes[DRBG_KEY_SIZE_BYTES];
   crypt_uword_t words[DRBG_KEY_SIZE_WORDS];
} DRBG_KEY;

typedef union
{
   BYTE bytes[DRBG_IV_SIZE_BYTES];
   crypt_uword_t words[DRBG_IV_SIZE_WORDS];
} DRBG_IV;
```

Part 4: Supporting Routines

Trusted Platform Module Library

40 BYTE bytes[DRBG_SEED_SIZE_BYTES];
41 crypt_uword_t words[DRBG_SEED_SIZE_WORDS];
42 } DRBG_SEED;
43 #define CTR_DRBG_MAX_REQUESTS_PER_RESEED ((UINT64)1 << 20)
44 #define CTR_DRBG_MAX_BYTES_PER_REQUEST (1 << 16)
45 # define CTR_DRBG_MIN_ENTROPY_INPUT_LENGTH DRBG_SEED_SIZE_BYTES
46 # define CTR_DRBG_MAX_ENTROPY_INPUT_LENGTH DRBG_SEED_SIZE_BYTES
47 # define CTR_DRBG_MAX_ADDITIONAL_INPUT_LENGTH DRBG_SEED_SIZE_BYTES
48 #define TESTING (1 << 0)
49 #define ENTROPY (1 << 1)
50 #define TESTED (1 << 2)
51 #define IsTestStateSet(BIT) ((g_cryptoSelfTestState.rng & BIT) != 0)
52 #define SetTestStateBit(BIT) (g_cryptoSelfTestState.rng |= BIT)
53 #define ClearTestStateBit(BIT) (g_cryptoSelfTestState.rng &= ~BIT)
54 #define IsSelfTest() IsTestStateSet(TESTING)
55 #define SetSelfTest() SetTestStateBit(TESTING)
56 #define ClearSelfTest() ClearTestStateBit(TESTING)
57 typedef struct {
58     UINT64      reseedCounter;
59     UINT32      magic;
60     DRBG_SEED   seed; // contains the key and IV for the counter mode DRBG
61     UINT32      lastValue[4]; // used when the TPM does continuous self-test
62 } DRBG_STATE, *pDRBG_STATE;
63 #define DRBG_MAGIC   ((UINT32) 0x47425244) // "DRBG" backwards so that it displays
64 typedef struct {
65     UINT64               counter;
66     UINT32               magic;
67     UINT32               limit;
68     TPM2B               *seed; // contains the key and IV for the counter mode DRBG
69 } KDF_STATE, *pKDF_STATE;
70 #define KDF_MAGIC    ((UINT32) 0x4048444a) // "KDF " backwards so that it displays
71 typedef struct {
72     UINT64        reseedCounter;
73     UINT32        magic;
74     UINT32        limit;
75     TPM2B        *seed;
76     const TPM2B   *label;
77     TPM2B        *context;
78     TPM_ALG_ID   hash;
79     TPM_ALG_ID   kdf;
80     UINT16       digestSize;
81     TPM2B_DIGEST residual;
82 } KDF_STATE, *pKDF_STATE;
83 #define KDF_MAGIC    ((UINT32) 0x40484444) // "KDF " backwards

Make sure that any other structures added to this union start with a 64-bit counter and a 32-bit magic number

86 typedef union
87 {
88     DRBG_STATE      drbg;
89     KDF_STATE       kdf;
90 } RAND_STATE;

This is the state used when the library uses a random number generator. A special function is installed for the library to call. That function picks up the state from this location and uses it for the generation of the random number.

91 extern RAND_STATE *s_random;

When instrumenting RSA key sieve
92  #if RSA_INSTRUMENT
93  #define PRIME_INDEX(x) ((x) == 512 ? 0 : (x) == 1024 ? 1 : 2)
94  # define INSTRUMENT_SET(a, b) ((a) = (b))
95  # define INSTRUMENT_ADD(a, b) (a) = (a) + (b)
96  # define INSTRUMENT_INC(a) (a) = (a) + 1
97  extern UINT32 PrimeIndex;
98  extern UINT32 failedAtIteration[10];
99  extern UINT32 PrimeCounts[3];
100 extern UINT32 MillerRabinTrials[3];
101 extern UINT32 totalFieldsSieved[3];
102 extern UINT32 bitsInFieldAfterSieve[3];
103 extern UINT32 emptyFieldsSieved[3];
104 extern UINT32 noPrimeFields[3];
105 extern UINT32 primesChecked[3];
106 extern UINT16 lastSievePrime;
107  #else
108  # define INSTRUMENT_SET(a, b)
109  # define INSTRUMENT_ADD(a, b)
110  # define INSTRUMENT_INC(a)
111  #endif
112  #endif // _CRYPT_RAND_H


10.1.5  CryptRsa.h

This file contains the RSA-related structures and defines.

```c
1 #ifndef CRYPT_RSA_H
2 #define _CRYPT_RSA_H
3
4 typedef struct
5 {
6     UINT32        exponent; // The public exponent pointer
7     TPM2B        *publicKey; // Pointer to the public modulus
8     TPM2B        *privateKey; // The private prime
9 ) RSA_KEY;

These values are used in the *bigNum* representation of various RSA values.

10 #define RSA_BITS            (MAX_RSA_KEY_BYTES * 8)
11 BN_TYPE(rsa, RSA_BITS);
12 #define BN_RSA(name)       BN_VAR(name, RSA_BITS)
13 #define BN_RSA_INITIALIZED(name, initializer)
14 \BN_INITIALIZED(name, RSA_BITS, initializer)
15 #define BN_PRIME(name)     BN_VAR(name, (RSA_BITS / 2))
16 BN_TYPE(prime, (RSA_BITS / 2));
17 #define BN_PRIME_INITIALIZED(name, initializer)
18 \BN_INITIALIZED(name, RSA_BITS / 2, initializer)
19
typedef struct privateExponent
20 {
21    #if CRT_FORMAT_RSA == NO
22        bn_rsa_t            D;
23    #else
24        bn_prime_t          Q;
25        bn_prime_t          dP;
26        bn_prime_t          dQ;
27        bn_prime_t          qInv;
28    #endif // CRT_FORMAT_RSA
29 } privateExponent_t;
30 #endif // _CRYPT_RSA_H
```
10.1.6 CryptTest.h

This file contains constant definitions used for self-test.

```
#ifndef _CRYPT_TEST_H
#define _CRYPT_TEST_H

This is the definition of a bit array with one bit per algorithm.

NOTE: Since bit numbering starts at zero, when ALG_LAST_VALUE is a multiple of 8, ALGORITHM_VECTOR will need to have byte for the single bit in the last byte. So, for example, when ALG_LAST_VECTOR is 8, ALGORITHM_VECTOR will need 2 bytes.

#define ALGORITHM_VECTOR_BYTES ((ALG_LAST_VALUE + 8) / 8)
typedef BYTE ALGORITHM_VECTOR[ALGORITHM_VECTOR_BYTES];

#ifdef TEST_SELF_TEST
LIB_EXPORT extern ALGORITHM_VECTOR LibToTest;
#endif

This structure is used to contain self-test tracking information for the cryptographic modules. Each of the major modules is given a 32-bit value in which it may maintain its own self test information. The convention for this state is that when all of the bits in this structure are 0, all functions need to be tested.

typedef struct
{
    UINT32      rng;
    UINT32      hash;
    UINT32      sym;
#if ALG_RSA
    UINT32      rsa;
#endif
#if ALG_ECC
    UINT32      ecc;
#endif
} CRYPTO_SELF_TEST_STATE;

This structure contains the self-test state values for the cryptographic modules.

extern CRYPTO_SELF_TEST_STATE g_cryptoSelfTestState;
#endif // _CRYPT_TEST_H
```
10.1.7 HashTestData.h

Hash Test Vectors

```c
TPM2B_TYPE (HASH_TEST_KEY, 128); // Twice the largest digest size
TPM2B_HASH_TEST_KEY c_hashTestKey = {{128, {
  0xa0,0xed,0x5c,0x9a,0xd2,0x4a,0x21,0x40,0x1a,0xd0,0x81,0x47,0x39,0x63,0xf9,0x50,
  0xdc,0x59,0x47,0x11,0x40,0x13,0x99,0x92,0xc0,0x72,0xa4,0x0f,0xe2,0x33,0xe4,0x63,
  0x9b,0xb6,0x76,0xc3,0xe0,0x6f,0x13,0x3e,0xcc,0x99,0x71,0xa5,0xc0,0xcf,0x9a,0x40,
  0xcf,0xdb,0x66,0x70,0x05,0x63,0x54,0x12,0x25,0xf4,0xe0,0x1b,0x23,0x35,0xe3,0x70,
  0x7d,0x19,0x5f,0x00,0xe4,0xf1,0x61,0x73,0x05,0xd8,0x58,0x7f,0x60,0x61,0x84,0x36,
  0xec,0xbe,0x96,0x1b,0x69,0x00,0xf0,0x9a,0x66,0xe3,0x26,0x73,0xd0,0x17,0x5b,0x33,
  0x14,0x41,0x9d,0x90,0xab,0x9d,0x6b,0x7d,0x48,0x99,0x25,0x93,0x29,0x14,0x2b,0xce,
  0x93,0x8d,0x8c,0xa5,0xe5,0xc6,0x57,0x8d,0x5b,0x57,0x20,0x1b,0x9f,0x2d,0xa5
}});
TPM2B_TYPE (HASH_TEST_DATA, 256); // Twice the largest block size
TPM2B_HASH_TEST_DATA c_hashTestData = {{256, {
  0x88,0xac,0xc3,0xe5,0x5f,0x66,0x9d,0x18,0x80,0x97,0x9a,0xc9,0xa4,0x08,0x90,0x98,
  0x0f,0x3a,0x59,0x92,0x4c,0x67,0x4e,0xb7,0x37,0xec,0x67,0x87,0xb6,0xbe,0x10,0xca,
  0x11,0x5b,0x4a,0xb0,0x45,0xc3,0x32,0x68,0x48,0x69,0xce,0x25,0xb1,0x8e,0xf0,0x44,
  0x79,0xb2,0x83,0xc8,0xb0,0xe2,0x63,0x94,0xa2,0x3c,0x59,0xe3,0xe6,0x64,0x2c,
  0x1f,0x8c,0x11,0x93,0x24,0xa3,0x17,0xc5,0x2f,0x37,0xcf,0x95,0x97,0x8e,0x63,0x39,
  0x68,0x5d,0xca,0xba,0x18,0x37,0x69,0x6e,0x4f,0x19,0xf9,0x8a,0xc0,0x68,0x87,0x3a,
  0x0b,0x31,0x42,0x04,0x05,0xf1,0xb5,0xe0,0x7e,0xe2,0x9a,0x24,0xb7,0x73,0x2c,0x8c,
  0x8e,0xb3,0x13,0x81,0x34,0xb9,0xb5,0xc1,0x17,0x37,0x39,0xf8,0x3e,0x84,0x4c,0x06,
  0x8a,0xb1,0x52,0x2f,0xe0,0xc9,0x9c,0x69,0x89,0xbc,0x85,0x9c,0x30,0x16,0x62,0x0c,
  0xe3,0x61,0xd4,0x0f,0x30,0x64,0x1b,0xca,0xc1,0x1b,0xda,0x1f,0xa2,0x22,0xe0,0xdf,
  0x52,0x2f,0x0b,0x4b,0x9f,0xe0,0x45,0x54,0xb9,0x17,0xb6,0xaf,0x6d,0x65,0xca,0x90,
  0x29,0x57,0x7b,0x70,0x50,0x94,0x5c,0x8e,0xf6,0x4e,0x21,0x8b,0xc6,0xb8,0x6a,0xbc,
  0xb9,0x64,0xd4,0xdf,0x38,0xd8,0xac,0xde,0xd8,0xb5,0x6d,0xcd,0x93,0xe8,
  0x28,0xa4,0xe2,0x5c,0x44,0xef,0xf0,0xe1,0x6f,0x38,0x1a,0xc3,0xe6,0xef,0xa2,0x9d,
  0xb9,0x8a,0x05,0xa9,0x45,0xb5,0xdb,0xb0,0x25,0x67,0xc9,0x86,0x7a,0xe8,0xe,
  0x51,0xcc,0xc3,0xd3,0xff,0x6e,0xf0,0xed,0xa3,0xae,0xf9,0x5d,0x33,0x70,0xf2,0x11
}});
#if ALG_SHA1 == YES
TPM2B_TYPE (SHA1, 20);
TPM2B_SHA1 c_SHA1_digest = {{20, {
  0x19,0xe6,0x3f,0x93,0x76,0x6d,0xf8,0x91,0x9c,0xe6,0xe5,0x57,0x53,0x77,0x01,0xb5,
  0x70,0x95,0xe5,0x40
}});
#endif
#if ALG_SHA256 == YES
TPM2B_TYPE (SHA256, 32);
TPM2B_SHA256 c_SHA256_digest = {{32, {
  0x64,0xe8,0xe0,0xc3,0xa9,0xa4,0x51,0x49,0x10,0x55,0x8d,0x31,0x71,0xe5,0x2f,0x69,
  0x3a,0xc6,0xc7,0x11,0x32,0x44,0x61,0xb6,0x34,0x39,0x57,0xb0,0xa8,0x75,0x86,0xb1
}});
#endif
#if ALG_SHA384 == YES
TPM2B_TYPE (SHA384, 48);
TPM2B_SHA384 c_SHA384_digest = {{48, {
  0x37,0x75,0x29,0xb5,0x20,0x15,0x6e,0xa3,0x7e,0xa3,0xd0,0xc9,0x80,0x8a,0x3a,0x3d,
  0xeb,0xe8,0xad,0x4e,0xc1,0x77,0x94,0x5a,0xaf,0xc6,0xd0,0xc1,0xfa,0x43,0x3f,0xc7,
  0xb8,0xf1,0x01,0xc0,0x60,0xbf,0xf2,0x87,0xe8,0x81,0x9e,0x51,0x97,0xa0,0x99,0x8d
}});
#endif
#if ALG_SHA512 == YES
TPM2B_TYPE (SHA512, 64);
TPM2B_SHA512 c_SHA512_digest = {{64, {
  0xe2,0x7b,0x10,0x3d,0x5e,0x48,0x58,0x44,0x67,0xac,0xa3,0x81,0x8c,0x1d,0xc5,0x71,
  0x66,0x92,0x89,0x89,0xaa,0xd4,0x35,0x51,0x60,0x37,0x31,0xe7,0x7b,0x93,0x0b,
  0x16,0x4d,0xb3,0xc8,0x34,0x98,0x3c,0xd3,0x53,0xde,0x5e,0xe8,0xc0,0xc6,0xaf,0xc9,
  0x24,0x2c,0xcc,0xed,0xdb,0x8a,0xb1,0x14,0x14,0x5a,0x95,0x80,0xde,0x66,0xbd
}});
#endif
```

Family “2.0”

TCG Public Review

Level 00 Revision 01.50
Copyright © TCG 2006-2019
September 18, 2018
10.1.8 KdfTestData.h

Hash Test Vectors

1 #define TEST_KDF_KEY_SIZE 20
2 TPM2B_TYPE(KDF_TEST_KEY, TEST_KDF_KEY_SIZE);
3 TPM2B_KDF_TEST_KEY c_kdfTestKeyIn = {{TEST_KDF_KEY_SIZE, {
4 0x27, 0x1F, 0xA0, 0x8B, 0xBD, 0xC5, 0x06, 0x0E, 0xC3, 0xDF,
5 0xA9, 0x28, 0xFF, 0x9B, 0x73, 0x12, 0x3A, 0x12, 0xDA, 0x0C }}};
6 TPM2B_TYPE(KDF_TEST_LABEL, 17);
7 TPM2B_KDF_TEST_LABEL c_kdfTestLabel = {{17, {
8 0x4B, 0x44, 0x46, 0x53, 0x45, 0x4C, 0x46, 0x54,
9 0x45, 0x53, 0x54, 0x4C, 0x41, 0x42, 0x45, 0x4C, 0x00 }}};
10 TPM2B_TYPE(KDF_TEST_CONTEXT, 8);
11 TPM2B_KDF_TEST_CONTEXT c_kdfTestContextU = {{8, {
12 0xCE, 0x24, 0x4F, 0x39, 0x5D, 0xCA, 0x73, 0x91 }}};
13 TPM2B_KDF_TEST_CONTEXT c_kdfTestContextV = {{8, {
14 0xDA, 0x50, 0x40, 0x31, 0xDD, 0xF1, 0x2E, 0x83 }}};
15 #if ALG_SHA512 == ALG_YES
16 TPM2B_KDF_TEST_KEY c_kdfTestKeyOut = {{20, {
17 0x8b, 0xe2, 0xc1, 0xb8, 0x5b, 0x78, 0x56, 0x9b, 0x9f, 0xa7,
18 0x59, 0xf5, 0x7c, 0x56, 0xda, 0xc4, 0x81, 0x0f, 0xda }}};
19 #define KDF_TEST_ALG TPM_ALG_SHA512
20 #elif ALG_SHA384 == ALG_YES
21 TPM2B_KDF_TEST_KEY c_kdfTestKeyOut = {{20, {
22 0x1d, 0xce, 0x70, 0xc9, 0x11, 0x3e, 0xb2, 0xdb, 0xa4, 0x7b,
23 0xda, 0x3f, 0xc7, 0x2b, 0xf4, 0x6f, 0x45, 0x0b, 0x93, 0x12 }}};
24 #define KDF_TEST_ALG TPM_ALG_SHA384
25 #elif ALG_SHA256 == ALG_YES
26 TPM2B_KDF_TEST_KEY c_kdfTestKeyOut = {{20, {
27 0xbb, 0x02, 0x59, 0xe1, 0xc8, 0xba, 0x60, 0x7e, 0x6a, 0x2c,
28 0xda, 0x04, 0x6b, 0x9a, 0x90, 0xe2, 0x9a, 0xde, 0x84, 0xc4 }}};
29 #define KDF_TEST_ALG TPM_ALG_SHA256
30 #elif ALG_SHA1 == ALG_YES
31 TPM2B_KDF_TEST_KEY c_kdfTestKeyOut = {{20, {
32 0x55, 0x05, 0xa7, 0x18, 0x4a, 0xa0, 0x74, 0x23, 0xc4, 0x7d,
33 0xae, 0x76, 0x6c, 0x26, 0xa2, 0x37, 0x7d, 0x7c, 0xf8, 0x51 }}};
34 #define KDF_TEST_ALG TPM_ALG_SHA1
35 #endif
Part 4: Supporting Routines

10.1.9

Trusted Platform Module Library

RsaTestData.h

RSA Test Vectors
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61

#define RSA_TEST_KEY_SIZE
256
typedef struct
{
UINT16
size;
BYTE
buffer[RSA_TEST_KEY_SIZE];
} TPM2B_RSA_TEST_KEY;
typedef TPM2B_RSA_TEST_KEY TPM2B_RSA_TEST_VALUE;
typedef struct
{
UINT16
size;
BYTE
buffer[RSA_TEST_KEY_SIZE / 2];
} TPM2B_RSA_TEST_PRIME;
const TPM2B_RSA_TEST_KEY
c_rsaPublicModulus = {256, {
0x91,0x12,0xf5,0x07,0x9d,0x5f,0x6b,0x1c,0x90,0xf6,0xcc,0x87,0xde,0x3a,0x7a,0x15,
0xdc,0x54,0x07,0x6c,0x26,0x8f,0x25,0xef,0x7e,0x66,0xc0,0xe3,0x82,0x12,0x2f,0xab,
0x52,0x82,0x1e,0x85,0xbc,0x53,0xba,0x2b,0x01,0xad,0x01,0xc7,0x8d,0x46,0x4f,0x7d,
0xdd,0x7e,0xdc,0xb0,0xad,0xf6,0x0c,0xa1,0x62,0x92,0x97,0x8a,0x3e,0x6f,0x7e,0x3e,
0xf6,0x9a,0xcc,0xf9,0xa9,0x86,0x77,0xb6,0x85,0x43,0x42,0x04,0x13,0x65,0xe2,0xad,
0x36,0xc9,0xbf,0xc1,0x97,0x84,0x6f,0xee,0x7c,0xda,0x58,0xd2,0xae,0x07,0x00,0xaf,
0xc5,0x5f,0x4d,0x3a,0x98,0xb0,0xed,0x27,0x7c,0xc2,0xce,0x26,0x5d,0x87,0xe1,0xe3,
0xa9,0x69,0x88,0x4f,0x8c,0x08,0x31,0x18,0xae,0x93,0x16,0xe3,0x74,0xde,0xd3,0xf6,
0x16,0xaf,0xa3,0xac,0x37,0x91,0x8d,0x10,0xc6,0x6b,0x64,0x14,0x3a,0xd9,0xfc,0xe4,
0xa0,0xf2,0xd1,0x01,0x37,0x4f,0x4a,0xeb,0xe5,0xec,0x98,0xc5,0xd9,0x4b,0x30,0xd2,
0x80,0x2a,0x5a,0x18,0x5a,0x7d,0xd4,0x3d,0xb7,0x62,0x98,0xce,0x6d,0xa2,0x02,0x6e,
0x45,0xaa,0x95,0x73,0xe0,0xaa,0x75,0x57,0xb1,0x3d,0x1b,0x05,0x75,0x23,0x6b,0x20,
0x69,0x9e,0x14,0xb0,0x7f,0xac,0xae,0xd2,0xc7,0x48,0x3b,0xe4,0x56,0x11,0x34,0x1e,
0x05,0x1a,0x30,0x20,0xef,0x68,0x93,0x6b,0x9d,0x7e,0xdd,0xba,0x96,0x50,0xcc,0x1c,
0x81,0xb4,0x59,0xb9,0x74,0x36,0xd9,0x97,0xdc,0x8f,0x17,0x82,0x72,0xb3,0x59,0xf6,
0x23,0xfa,0x84,0xf7,0x6d,0xf2,0x05,0xff,0xf1,0xb9,0xcc,0xe9,0xa2,0x82,0x01,0xfb}};
const TPM2B_RSA_TEST_PRIME
c_rsaPrivatePrime = {RSA_TEST_KEY_SIZE / 2, {
0xb7,0xa0,0x90,0xc7,0x92,0x09,0xde,0x71,0x03,0x37,0x4a,0xb5,0x2f,0xda,0x61,0xb8,
0x09,0x1b,0xba,0x99,0x70,0x45,0xc1,0x0b,0x15,0x12,0x71,0x8a,0xb3,0x2a,0x4d,0x5a,
0x41,0x9b,0x73,0x89,0x80,0x0a,0x8f,0x18,0x4c,0x8b,0xa2,0x5b,0xda,0xbd,0x43,0xbe,
0xdc,0x76,0x4d,0x71,0x0f,0xb9,0xfc,0x7a,0x09,0xfe,0x4f,0xac,0x63,0xd9,0x2e,0x50,
0x3a,0xa1,0x37,0xc6,0xf2,0xa1,0x89,0x12,0xe7,0x72,0x64,0x2b,0xba,0xc1,0x1f,0xca,
0x9d,0xb7,0xaa,0x3a,0xa9,0xd3,0xa6,0x6f,0x73,0x02,0xbb,0x85,0x5d,0x9a,0xb9,0x5c,
0x08,0x83,0x22,0x20,0x49,0x91,0x5f,0x4b,0x86,0xbc,0x3f,0x76,0x43,0x08,0x97,0xbf,
0x82,0x55,0x36,0x2d,0x8b,0x6e,0x9e,0xfb,0xc1,0x67,0x6a,0x43,0xa2,0x46,0x81,0x71}};
const BYTE
c_RsaTestValue[RSA_TEST_KEY_SIZE] = {
0x2a,0x24,0x3a,0xbb,0x50,0x1d,0xd4,0x2a,0xf9,0x18,0x32,0x34,0xa2,0x0f,0xea,0x5c,
0x91,0x77,0xe9,0xe1,0x09,0x83,0xdc,0x5f,0x71,0x64,0x5b,0xeb,0x57,0x79,0xa0,0x41,
0xc9,0xe4,0x5a,0x0b,0xf4,0x9f,0xdb,0x84,0x04,0xa6,0x48,0x24,0xf6,0x3f,0x66,0x1f,
0xa8,0x04,0x5c,0xf0,0x7a,0x6b,0x4a,0x9c,0x7e,0x21,0xb6,0xda,0x6b,0x65,0x9c,0x3a,
0x68,0x50,0x13,0x1e,0xa4,0xb7,0xca,0xec,0xd3,0xcc,0xb2,0x9b,0x8c,0x87,0xa4,0x6a,
0xba,0xc2,0x06,0x3f,0x40,0x48,0x7b,0xa8,0xb8,0x2c,0x03,0x14,0x33,0xf3,0x1d,0xe9,
0xbd,0x6f,0x54,0x66,0xb4,0x69,0x5e,0xbc,0x80,0x7c,0xe9,0x6a,0x43,0x7f,0xb8,0x6a,
0xa0,0x5f,0x5d,0x7a,0x20,0xfd,0x7a,0x39,0xe1,0xea,0x0e,0x94,0x91,0x28,0x63,0x7a,
0xac,0xc9,0xa5,0x3a,0x6d,0x31,0x7b,0x7c,0x54,0x56,0x99,0x56,0xbb,0xb7,0xa1,0x2d,
0xd2,0x5c,0x91,0x5f,0x1c,0xd3,0x06,0x7f,0x34,0x53,0x2f,0x4c,0xd1,0x8b,0xd2,0x9e,
0xdc,0xc3,0x94,0x0a,0xe1,0x0f,0xa5,0x15,0x46,0x2a,0x8e,0x10,0xc2,0xfe,0xb7,0x5e,
0x2d,0x0d,0xd1,0x25,0xfc,0xe4,0xf7,0x02,0x19,0xfe,0xb6,0xe4,0x95,0x9c,0x17,0x4a,
0x9b,0xdb,0xab,0xc7,0x79,0xe3,0x5e,0x40,0xd0,0x56,0x6d,0x25,0x0a,0x72,0x65,0x80,
0x92,0x9a,0xa8,0x07,0x70,0x32,0x14,0xfb,0xfe,0x08,0xeb,0x13,0xb4,0x07,0x68,0xb4,
0x58,0x39,0xbe,0x8e,0x78,0x3a,0x59,0x3f,0x9c,0x4c,0xe9,0xa8,0x64,0x68,0xf7,0xb9,
0x6e,0x20,0xf5,0xcb,0xca,0x47,0xf2,0x17,0xaa,0x8b,0xbc,0x13,0x14,0x84,0xf6,0xab};
const TPM2B_RSA_TEST_VALUE
c_RsaepKvt = {RSA_TEST_KEY_SIZE, {
0x73,0xbd,0x65,0x49,0xda,0x7b,0xb8,0x50,0x9e,0x87,0xf0,0x0a,0x8a,0x9a,0x07,0xb6,
0x00,0x82,0x10,0x14,0x60,0xd8,0x01,0xfc,0xc5,0x18,0xea,0x49,0x5f,0x13,0xcf,0x65,
0x66,0x30,0x6c,0x60,0x3f,0x24,0x3c,0xfb,0xe2,0x31,0x16,0x99,0x7e,0x31,0x98,0xab,
0x93,0xb8,0x07,0x53,0xcc,0xdb,0x7f,0x44,0xd9,0xee,0x5d,0xe8,0x5f,0x97,0x5f,0xe8,
0x1f,0x88,0x52,0x24,0x7b,0xac,0x62,0x95,0xb7,0x7d,0xf5,0xf8,0x9f,0x5a,0xa8,0x24,

Family “2.0”
Level 00 Revision 01.50

TCG Public Review
Copyright © TCG 2006-2019

Page 475
September 18, 2018


const TPM2B_RSA_TEST_VALUE c_RaesKvt = {RSA_TEST_KEY_SIZE, 
0x29,0xa4,0x2f,0xbb,0x8a,0x14,0x05,0x1e,0x3c,0x72,0x76,0x77,0x38,0xe7,0x73,0xe3, 
0x6e,0x24,0x4b,0x38,0xd2,0x1a,0xcf,0x23,0x58,0x78,0x36,0x82,0x23,0xe6,0x6b,0xef, 
0x2c,0x3d,0xf2,0x8e,0xd6,0x36,0x87,0xe8,0x78,0x9b,0x27,0x39,0xc0,0xd6,0xe0,0xe4, 
0x0b,0xfc,0x51,0x2f,0x18,0x0f,0x2e,0x4b,0x77,0x3e,0xc9,0xf3,0x3e,0x2a,0xc0,0x76, 
0x9f,0x4f,0x1d,0x5c,0x5d,0xd5,0x80,0x17,0x44,0x76,0x09,0x7a,0xc7,0x3e,0x3a,0x3e, 
0x6f,0x6f,0x3a,0x3f,0x8b,0x32,0x0d,0xd1,0x7e,0x80,0xe2,0xf7,0x4f,0xed,0xda,0x87, 
0x11,0x9c,0xc3,0xe7,0x85,0x18,0x58,0x08,0xb2,0x23,0x37,0x7e,0x9d,0x00,0xda, 
0x79,0xe2,0x05,0x0d,0xf5,0x4f,0xe0,0xf0,0xb8,0x4b,0x74,0xe8,0x89,0x01,0xb3,0xc3, 
0x0d,0xb6,0x7b,0x90,0xb1,0x13,0x77,0x30,0x4d,0x91,0x46,0x57,0xe2,0x5f,0x36, 
0x6e,0xf6,0x35,0xf9,0x6e,0xf6,0x35,0xf9,0x6e,0xf6,0x35,0xf9,0x6e,0xf6,0x35,0xf9,0x6e, 
0x0d,0x3e,0x7d,0x0c,0x5c,0x34,0xf0,0x11,0xa2,0x12,0xa7,0xe4,0x3a,0xd8,0xe0,0xe4, 
0xcf,0xf4,0x85,0x63,0x77,0x6c,0x33,0x34,0xe7,0x27,0xe4,0xe9,0x91,0xb0,0xe4,0x8b, 
0xf0,0xb0,0x79,0x7a,0xc9,0xa7,0x9a,0xc0,0x04,0x73,0xb8,0x95,0xd7,0x31,0x54, 
0x3b,0x56,0xec,0x52,0x15,0xd7,0x3e,0x62,0xf5,0x82,0x99,0x3e,0xa2,0xc0,0x4b,0xe2, 
0x06,0xe3,0x56,0xd3,0x7f,0x12,0x8b,0x2d,0xc5,0xb9,0x3b,0x68,0x56,0x73,0x70, 
0x32,0x6b,0x6b,0x65,0x25,0x76,0x45,0x6c,0x45,0xf1,0x6c,0x59,0xf3,0x94,0xa7,0x15};

const TPM2B_RSA_TEST_VALUE c_RaessKvt = {RSA_TEST_KEY_SIZE, 
0x01,0xefe,0xd5,0xb3,0x60,0xb1,0x90,0x2c,0x6d,0xf7,0x26,0xb7,0x8f,0xb1,0xd7, 
0x80,0xbf,0x83,0x60,0xad,0x8b,0x82,0x7b,0x85,0x38,0xa0,0x35,0x7d,0x3b,0x51, 
0xb8,0x7b,0x81,0x83,0xe6,0xff,0x76,0xe6,0xf0,0x59,0x34,0xd9,0x4b,0x89,0xe3, 
0x85,0x6f,0x5e,0x01,0x10,0x56,0xe6,0x93,0x06,0x8f,0x3d,0xc9,0x41,0xb9,0xc9,0xdf, 
0x59,0xa8,0xc3,0x1d,0xeb,0x96,0x4a,0x59,0x80,0x3c,0x90,0x3a,0x59,0x56,0x4c, 
0x44,0x6d,0xeb,0xdc,0x73,0xc0,0xc1,0xe0,0xb8,0x41,0xbf,0x89,0xc8,0x03,0x69,0x4c, 
0xae,0xf3,0xc1,0xc5,0xc7,0xe7,0x7d,0xa7,0x83,0x39,0x70,0x2a,0x6b,0x83,0xc0, 
0x72,0x4f,0x2f,0x16,0x15,0x68,0x69,0x5a,0x1d,0xfd, 
0x2c,0xa3,0x0f,0x67,0x1e,0x77,0x12,0x9e,0x83,0xdb,0xba,0xff,0x96,0xc9,0x1f,0xe4, 
0x32,0xf4,0xa7,0xb3,0x3f,0x7d,0x61,0xbb,0x9a,0x27,0xad,0xf0,0x2f,0x30,0xc4,0x70};

const TPM2B_RSA_TEST_VALUE c_RaesssKvt = {RSA_TEST_KEY_SIZE, 
0x67,0x4e,0xdd,0xc2,0xd2,0x6d,0xe0,0x03,0xc4,0xc2,0x41,0xd3,0xd4,0x61,0x30,0xd0,
```
#define ALG_SHA256_VALUE == DEFAULT_TEST_HASH

const TPM2B_RSA_TEST_VALUE c_OaepKvt = {RSA_TEST_KEY_SIZE, {0x33, 0x20, 0x6e, 0x21, 0xc3, 0xf6, 0xcd, 0xf8, 0x17, 0x3d, 0x2f, 0xe9, 0x05, 0x14, 0x8c, 0x7c, 0xf3, 0x2b, 0x85, 0x6a, 0x43, 0x84, 0x61, 0x4c, 0x59, 0x63, 0x3b, 0x4b, 0x2a, 0x00, 0x26, 0x4b, 0x2a, 0x00, 0x26, 0x4b, 0x8a, 0xe4, 0x99, 0x09, 0xe0, 0x1d, 0xf6, 0x5a, 0x6b, 0x17, 0x42, 0x58, 0x98, 0xd4, 0xe2, 0x88, 0x9c, 0x58, 0x0f, 0xec, 0x9f, 0x0e, 0x44, 0xcf, 0xa4, 0xb7, 0x5d, 0xb4, 0xd0, 0x71, 0xe9, 0xd6, 0x9f, 0xf4, 0x47, 0x70, 0x60, 0xf8, 0xd7, 0x29, 0x1e, 0x91, 0x36, 0x3e, 0x4a, 0x94, 0x1e, 0xb7, 0x5d, 0xb4, 0xd0, 0x71, 0xe9, 0xd6, 0x9f, 0xf4, 0x47, 0x70, 0x60, 0xf8, 0xd7, 0x29, 0x1e, 0x91, 0x36, 0x3e, 0x4a, 0x94, 0x1e, 0xb7, 0x5d, 0xb4, 0xd0, 0x71, 0xe9, 0xd6, 0x9f, 0xf4, 0x47, 0x70, 0x60, 0xf8, 0xd7, 0x29, 0x1e, 0x91, 0x36, 0x3e, 0x4a, 0x94, 0x1e, 0xb7, 0x5d, 0xb4, 0xd0, 0x71, 0xe9, 0xd6, 0x9f, 0xf4, 0x47, 0x70, 0x60, 0xf8, 0xd7, 0x29, 0x1e, 0x91, 0x36, 0x3e, 0x4a, 0x94, 0x1e, 0xb7, 0x5d, 0xb4, 0xd0, 0x71, 0xe9, 0xd6, 0x9f, 0xf4, 0x47, 0x70, 0x60, 0xf8, 0xd7, 0x29, 0x1e, 0x91, 0x36, 0x3e, 0x4a, 0x94, 0x1e, 0xb7, 0x5d, 0xb4, 0xd0, 0x71, 0xe9, 0xd6, 0x9f, 0xf4, 0x47, 0x70, 0x60, 0xf8, 0xd7, 0x29, 0x1e, 0x91, 0x36, 0x3e, 0x4a, 0x94, 0x1e, 0xb7, 0x5d, 0xb4, 0xd0, 0x71, 0xe9, 0xe2, 0xe1, 0x4f, 0x9f, 0x44, 0xc8, 0x46, 0x9f, 0x0e, 0x8a, 0x16, 0x8a, 0xa0, 0x55, 0x6e, 0x66, 0x3a, 0x23, 0xe0, 0x47, 0x57, 0x2c, 0x25, 0xbc, 0xa5, 0xe2, 0xc7, 0xe3}};
```
0x0b,0x65,0x6b,0x2b,0x6d,0x55,0x95,0x85,0x44,0x8b,0x12,0x05,0x3f,0x4b,0xd4,0x8e,
0x3d,0x68,0x2d,0x29,0x9c,0x05,0x79,0xd6,0xfc,0x72,0x90,0x6a,0xb,0x46,0x38,0x81};

const TPM2B_RSA_TEST_VALUE c_ReassKvt = {RSA_TEST_KEY_SIZE, {
  0x8a,0x8b,0x6b,0xe4,0x02,0xf7,0xdd,0x45,0x2a,0xcc,0x2b,0x6b,0x8c,0xe0,0x9a,
  0x92,0x4f,0x9b,0xc5,0xe4,0xb8,0xb9,0xb0,0xd9,0x87,0x8c,0xcb,0xf0,0xb0,0x59,
  0xa5,0x92,0x21,0xa0,0x7a,0x5c,0xed,0xa8,0xe6,0x22,0x29,0x46,0xc7,0x86,0x37,
  0x4b,0x1b,0xe9,0x93,0xc8,0x4c,0x17,0xa7,0xae,0x59,0x91,0xf8,0x83,0x84,0xc4,
  0x8c,0x38,0xc2,0x35,0xe0,0xe7,0x50,0x67,0x76,0x7e,0x03,0x3c,0xce,0xf6,0x0d,0x0a,0x5c,
  0x2f,0xa0,0x80,0x28,0xd3,0xc5,0x7d,0x2d,0x1a,0xb,0x96,0xd6,0x5e,0x98,0x05,0x8c,
  0x4d,0x2a,0x1f,0x8c,0xb6,0xfb,0xb1,0xc1,0xe9,0xcb,0x38,0x27,0x60,0x64,0x17,0xca,
  0xf4,0xb8,0x61,0x67,0x2b,0x0a,0x20,0x9d,0x40,0xa2,0x1c,0xfd,0x55,0x40,0x4b,0x95,
  0x39,0x52,0x18,0x3b,0xa4,0xb4,0xe8,0x83,0x4b,0x7c,0x47,0xfb,0xed,0x06,0x9c,0xc3,
  0x4f,0xba,0x81,0xd6,0xb7,0x31,0xc3,0xc5,0x2c,0x5d,0x25,0x2b,0x95,0x77,0xa0,0x8f,
  0x46,0xcf,0xfb,0x59,0xb8,0xe4,0xd7,0x2e,0xd5,0x0e,0x7f,0x5a,0xb9,0xe5,0x2b,0x50,0x19,
  0xf5,0x7f,0x84,0x6f,0xc3,0xed,0xf8,0x24,0xb0,0x43,0xd1,0xb4,0x7a,0x89,0xe4,0x39,
  0x3e,0x2e,0xa5,0x16,0xa5,0x28,0xee,0x96,0x84,0xe3,0x16,0xd6,0x5f,0xe4,0xb0,0x7d,
  0x94,0x16,0x1b,0xc8,0xf9,0xaa,0x9b,0xc0,0x49,0x42,0x4c,0xe3,0x62,0x0f,0xe3,0xa2,
  0x20,0x33,0x5e,0xa6,0xdd,0xda,0x15,0x2d,0xb7,0xcc,0xda,0xff,0xb1,0x0b,0x45,0x7b,
  0x2d,0x9a,0x44,0x29,0xa9,0x73,0xe9,0x9a,0x4d,0x99,0x8d,0xcac,0xa1,0x98,0x2c,0x2d};

#endif // SHA256

#if ALG_SHA384 VALUE == DEFAULT_TEST_RASH

const TPM2B_RSA_TEST_VALUE c_OaepKvt = {RSA_TEST_KEY_SIZE, {
  0x87,0x47,0x7e,0x09,0xc4,0x13,0x45,0x0b,0x3f,0x3a,0x82,0x6d,0xe3,0xd1,0xe8,0x83,
  0x3f,0xda,0x6c,0xbe,0xbe,0xcd,0x00,
  0x4d,0x5f,0x7f,0xf8,0xb9,0xd1,0x6f,0x8d,0x3e,0x45,0x9d,0x86,0x12,0x3f,
  0x29,0x91,0x96,0x05,0x3c,0xfd,0x59,0x3b,0x7f,0x29,0x2b,0x03,
  0x46,0xc5,0x3b,0xd8,0xf8,0xa2,0x41,0x6a,0x60,0xb8,0xe9,0x5e,0x7f,0x20,0x16,0xe3,
  0x69,0xb6,0x2d,0x92,0xf6,0xa0,0x27,0x74,0x88,0x5d,0xc7,0x6a,0xd1,0xff,0x3e,0x45,
  0x80,0x5b,0x39,0xdf,0x56,0xb0,0x91,0xb0,0x80,0x0e,0x6c,0xa8,0x3c,0x78,0xef,0x34,
  0xc0,0x2e,0x88,0x5f,0xfb,0x47,0x98,0x5d,0x57,0xe8,0x3a,0xb9,0xff,0xr,0x20,0x04,
  0x6e,0xfa,0x14,0xc1,0xb9,0x68,0x15,0xc5,0x12,0xe8,0xa8,0xbe,0xea,0x88,0xd8,0x9b,
  0xc2,0x57,0x2d,0xb6,0x3f,0x5b,0x6c,0xc0,0x2c,0x2f,0xa9,0x50,0x0b,0xda,0x5b,0xbb,
  0x1f,0xe4,0xb2,0x1a,0x69,0x0d,0xd9,0x98,0x8e,0xdb,0x85,0x61,0x70,0x20,
  0x82,0x91,0x26,0x87,0x08,0x6a,0x3d,0xb1,0x94,0x4d,0x0d,0x33,0x84,0xad,0xbb7};

#endif // SHA256
Part 4: Supporting Routines
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325

Trusted Platform Module Library

0x43,0x3a,0x0d,0xb1,0x1b,0x10,0x66,0x81,0xe2,0x0d,0xe7,0xd1,0xca,0x85,0xa7,0x50,
0x82,0x2d,0xbf,0xed,0xcf,0x43,0x6d,0xdb,0x2c,0x7b,0x73,0x20,0xfe,0x73,0x3f,0x19,
0xc6,0xdb,0x69,0xb8,0xc3,0xd3,0xf4,0xe5,0x64,0xf8,0x36,0x8e,0xd5,0xd8,0x09,0x2a,
0x5f,0x26,0x70,0xa1,0xd9,0x5b,0x14,0xf8,0x22,0xe9,0x9d,0x22,0x51,0xf4,0x52,0xc1,
0x6f,0x53,0xf5,0xca,0x0d,0xda,0x39,0x8c,0x29,0x42,0xe8,0x58,0x89,0xbb,0xd1,0x2e,
0xc5,0xdb,0x86,0x8d,0xaf,0xec,0x58,0x36,0x8d,0x8d,0x57,0x23,0xd5,0xdd,0xb9,0x24}};
const TPM2B_RSA_TEST_VALUE
c_RsassaKvt = {RSA_TEST_KEY_SIZE, {
0x39,0x10,0x58,0x7d,0x6d,0xa8,0xd5,0x90,0x07,0xd6,0x2b,0x13,0xe9,0xd8,0x93,0x7e,
0xf3,0x5d,0x71,0xe0,0xf0,0x33,0x3a,0x4a,0x22,0xf3,0xe6,0x95,0xd3,0x8e,0x8c,0x41,
0xe7,0xb3,0x13,0xde,0x4a,0x45,0xd3,0xd1,0xfb,0xb1,0x3f,0x9b,0x39,0xa5,0x50,0x58,
0xef,0xb6,0x3a,0x43,0xdd,0x54,0xab,0xda,0x9d,0x32,0x49,0xe4,0x57,0x96,0xe5,0x1b,
0x1d,0x8f,0x33,0x8e,0x07,0x67,0x56,0x14,0xc1,0x18,0x78,0xa2,0x52,0xe6,0x2e,0x07,
0x81,0xbe,0xd8,0xca,0x76,0x63,0x68,0xc5,0x47,0xa2,0x92,0x5e,0x4c,0xfd,0x14,0xc7,
0x46,0x14,0xbe,0xc7,0x85,0xef,0xe6,0xb8,0x46,0xcb,0x3a,0x67,0x66,0x89,0xc6,0xee,
0x9d,0x64,0xf5,0x0d,0x09,0x80,0x9a,0x6f,0x0e,0xeb,0xe4,0xb9,0xe9,0xab,0x90,0x4f,
0xe7,0x5a,0xc8,0xca,0xf6,0x16,0x0a,0x82,0xbd,0xb7,0x76,0x59,0x08,0x2d,0xd9,0x40,
0x5d,0xaa,0xa5,0xef,0xfb,0xe3,0x81,0x2c,0x2c,0x5c,0xa8,0x16,0xbd,0x63,0x20,0xc2,
0x4d,0x3b,0x51,0xaa,0x62,0x1f,0x06,0xe5,0xbb,0x78,0x44,0x04,0x0c,0x5c,0xe1,0x1b,
0x6b,0x9d,0x21,0x10,0xaf,0x48,0x48,0x98,0x97,0x77,0xc2,0x73,0xb4,0x98,0x64,0xcc,
0x94,0x2c,0x29,0x28,0x45,0x36,0xd1,0xc5,0xd0,0x2f,0x97,0x27,0x92,0x65,0x22,0xbb,
0x63,0x79,0xea,0xf5,0xff,0x77,0x0f,0x4b,0x56,0x8a,0x9f,0xad,0x1a,0x97,0x67,0x39,
0x69,0xb8,0x4c,0x6c,0xc2,0x56,0xc5,0x7a,0xa8,0x14,0x5a,0x24,0x7a,0xa4,0x6e,0x55,
0xb2,0x86,0x1d,0xf4,0x62,0x5a,0x2d,0x87,0x6d,0xde,0x99,0x78,0x2d,0xef,0xd7,0xdc}};
#endif // SHA384
#if ALG_SHA512_VALUE == DEFAULT_TEST_HASH
const TPM2B_RSA_TEST_VALUE
c_OaepKvt = {RSA_TEST_KEY_SIZE, {
0x48,0x45,0xa7,0x70,0xb2,0x41,0xb7,0x48,0x5e,0x79,0x8c,0xdf,0x1c,0xc6,0x7e,0xbb,
0x11,0x80,0x82,0x52,0xbf,0x40,0x3d,0x90,0x03,0x6e,0x20,0x3a,0xb9,0x65,0xc8,0x51,
0x4c,0xbd,0x9c,0xa9,0x43,0x89,0xd0,0x57,0x0c,0xa3,0x69,0x22,0x7e,0x82,0x2a,0x1c,
0x1d,0x5a,0x80,0x84,0x81,0xbb,0x5e,0x5e,0xd0,0xc1,0x66,0x9a,0xac,0x00,0xba,0x14,
0xa2,0xe9,0xd0,0x3a,0x89,0x5a,0x63,0xe2,0xec,0x92,0x05,0xf4,0x47,0x66,0x12,0x7f,
0xdb,0xa7,0x3c,0x5b,0x67,0xe1,0x55,0xca,0x0a,0x27,0xbf,0x39,0x89,0x11,0x05,0xba,
0x9b,0x5a,0x9b,0x65,0x44,0xad,0x78,0xcf,0x8f,0x94,0xf6,0x9a,0xb4,0x52,0x39,0x0e,
0x00,0xba,0xbc,0xe0,0xbd,0x6f,0x81,0x2d,0x76,0x42,0x66,0x70,0x07,0x77,0xbf,0x09,
0x88,0x2a,0x0c,0xb1,0x56,0x3e,0xee,0xfd,0xdc,0xb6,0x3c,0x0d,0xc5,0xa4,0x0d,0x10,
0x32,0x80,0x3e,0x1e,0xfe,0x36,0x8f,0xb5,0x42,0xc1,0x21,0x7b,0xdf,0xdf,0x4a,0xd2,
0x68,0x0c,0x01,0x9f,0x4a,0xfd,0xd4,0xec,0xf7,0x49,0x06,0xab,0xed,0xc6,0xd5,0x1b,
0x63,0x76,0x38,0xc8,0x6c,0xc7,0x4f,0xcb,0x29,0x8a,0x0e,0x6f,0x33,0xaf,0x69,0x31,
0x8e,0xa7,0xdd,0x9a,0x36,0xde,0x9b,0xf1,0x0b,0xfb,0x20,0xa0,0x6d,0x33,0x31,0xc9,
0x9e,0xb4,0x2e,0xc5,0x40,0x0e,0x60,0x71,0x36,0x75,0x05,0xf9,0x37,0xe0,0xca,0x8e,
0x8f,0x56,0xe0,0xea,0x9b,0xeb,0x17,0xf3,0xca,0x40,0xc3,0x48,0x01,0xba,0xdc,0xc6,
0x4b,0x2b,0x5b,0x7b,0x5c,0x81,0xa6,0xbb,0xc7,0x43,0xc0,0xbe,0xc0,0x30,0x7b,0x55}};
const TPM2B_RSA_TEST_VALUE
c_RsaesKvt = {RSA_TEST_KEY_SIZE, {
0x74,0x83,0xfa,0x52,0x65,0x50,0x68,0xd0,0x82,0x05,0x72,0x70,0x78,0x1c,0xac,0x10,
0x23,0xc5,0x07,0xf8,0x93,0xd2,0xeb,0x65,0x87,0xbb,0x47,0xc2,0xfb,0x30,0x9e,0x61,
0x4c,0xac,0x04,0x57,0x5a,0x7c,0xeb,0x29,0x08,0x84,0x86,0x89,0x1e,0x8f,0x07,0x32,
0xa3,0x8b,0x70,0xe7,0xa2,0x9f,0x9c,0x42,0x71,0x3d,0x23,0x59,0x82,0x5e,0x8a,0xde,
0xd6,0xfb,0xd8,0xc5,0x8b,0xc0,0xdb,0x10,0x38,0x87,0xd3,0xbf,0x04,0xb0,0x66,0xb9,
0x85,0x81,0x54,0x4c,0x69,0xdc,0xba,0x78,0xf3,0x4a,0xdb,0x25,0xa2,0xf2,0x34,0x55,
0xdd,0xaa,0xa5,0xc4,0xed,0x55,0x06,0x0e,0x2a,0x30,0x77,0xab,0x82,0x79,0xf0,0xcd,
0x9d,0x6f,0x09,0xa0,0xc8,0x82,0xc9,0xe0,0x61,0xda,0x40,0xcd,0x17,0x59,0xc0,0xef,
0x95,0x6d,0xa3,0x6d,0x1c,0x2b,0xee,0x24,0xef,0xd8,0x4a,0x55,0x6c,0xd6,0x26,0x42,
0x32,0x17,0xfd,0x6a,0xb3,0x4f,0xde,0x07,0x2f,0x10,0xd4,0xac,0x14,0xea,0x89,0x68,
0xcc,0xd3,0x07,0xb7,0xcf,0xba,0x39,0x20,0x63,0x20,0x7b,0x44,0x8b,0x48,0x60,0x5d,
0x3a,0x2a,0x0a,0xe9,0x68,0xab,0x15,0x46,0x27,0x64,0xb5,0x82,0x06,0x29,0xe7,0x25,
0xca,0x46,0x48,0x6e,0x2a,0x34,0x57,0x4b,0x81,0x75,0xae,0xb6,0xfd,0x6f,0x51,0x5f,
0x04,0x59,0xc7,0x15,0x1f,0xe0,0x68,0xf7,0x36,0x2d,0xdf,0xc8,0x9d,0x05,0x27,0x2d,
0x3f,0x2b,0x59,0x5d,0xcb,0xf3,0xc4,0x92,0x6e,0x00,0xa8,0x8d,0xd0,0x69,0xe5,0x59,
0xda,0xba,0x4f,0x38,0xf5,0xa0,0x8b,0xf1,0x73,0xe9,0x0d,0xee,0x64,0xe5,0xa2,0xd8}};
const TPM2B_RSA_TEST_VALUE
c_RsapssKvt = {RSA_TEST_KEY_SIZE, {
0x1b,0xca,0x8b,0x18,0x15,0x3b,0x95,0x5b,0x0a,0x89,0x10,0x03,0x7f,0x7c,0xa0,0xc9,
0x66,0x57,0x86,0x6a,0xc9,0xeb,0x82,0x71,0xf3,0x8d,0x6f,0xa9,0xa4,0x2d,0xd0,0x22,
0xdf,0xe9,0xc6,0x71,0x5b,0xf4,0x27,0x38,0x5b,0x2c,0x8a,0x54,0xcc,0x85,0x11,0x69,
0x6d,0x6f,0x42,0xe7,0x22,0xcb,0xd6,0xad,0x1a,0xc5,0xab,0x6a,0xa5,0xfc,0xa5,0x70,
0x72,0x4a,0x62,0x25,0xd0,0xa2,0x16,0x61,0xab,0xac,0x31,0xa0,0x46,0x24,0x4f,0xdd,
0x9a,0x36,0x55,0xb6,0x00,0x9e,0x23,0x50,0x0d,0x53,0x01,0xb3,0x46,0x56,0xb2,0x1d,

Family “2.0”
Level 00 Revision 01.50

TCG Public Review
Copyright © TCG 2006-2019

Page 479
September 18, 2018


const TPM2B_RSA_TEST_VALUE c_RsassaKvt = {RSA_TEST_KEY_SIZE, {
    0x05,0x55,0x00,0x62,0x01,0xc6,0x04,0x31,0x55,0x73,0x3f,0x02a,0x7f,0xda,0x0f,0xc1,
    0x2b,0xebe,0x88,0xc8,0xb2,0xab,0x6c,0x26,0xd2,0x2d,0x9c,0xe2,0x6d,0x14,0x9a,
    0xc8,0x22,0x5d,0x58,0x03,0xb1,0x46,0x14,0xa5,0xd4,0xb2,0x7f,0xe1,0x14,0x7e,
    0xe3,0x3d,!x8,0x3a,0x95,0x79,0x13,0xe7,0xe2,0x2a,0xb9,0x3e,0x8f,0xa,0x01,
    0x01,0x7c,0x03,0x12,0xa,0x2a,0xb4,0x39,0x98,0x6d,0x0ff,0x0a,0x6c,0x39,0x2d,
    0xdc,0xc7,0x77,0x62,0x10,0xa6,0xc,0xe2,0x71,0xe8,0x97,0x00,0x87,0x5b,0x0e,
    0x20,0x00,0x3f,0x18,0x63,0x83,0xf0,0xe4,0x0a,0x64,0x8c,0xe9,0x8c,0x91,0xe7,0x89,
    0x04,0x64,0x2c,0x8b,0x41,0xc8,0xac,0xf6,0x5a,0x75,0xe6,0xa5,0x76,0xe4,0xcb,0xa,0x5,
    0x33,0x8b,0x07,0xc9,0x73,0x0f,0x45,0x2a,0xc1,0xe3,0xe6,0xe7,0x21,0x66,
    0x1c,0x8b,0x8f,0xe6,0x3e,0x39,0x9a,0x2b,0xe2,0x8f,0xe0,0xe9c,0xb4,0x85,0x89,0x33,
    0x2a,0x0c,0xc8,0x5d,0x58,0xe1,0x89,0x12,0x8f,0x4d,0x2f,0x1f,0x99,0x0c,0x3e,
    0xda,0xb2,0xe8,0x5f,0x8b,0xe1,0x4b,0x88,0xed,0x3d,0xa8,0xda,0xe8,0x4d,0x45,
    0xb,0x8b,0x41,0xc8,0xac,0xf6,0x5a,0x75,0xe6,0xa5,0x76,0xe4,0xcb,0xa,
    0x33,0x8b,0x07,0xc9,0x73,0x0f,0x45,0x2a,0xc1,0xe3,0xe6,0xe7,0x21,0x66,
    0x1c,0x8b,0x8f,0xe6,0x3e,0x39,0x9a,0x2b,0xe2,0x8f,0xe0,0xe9c,0xb4,0x85,0x89,0x33,
    0x2a,0x0c,0xc8,0x5d,0x58,0xe1,0x89,0x12,0x8f,0x4d,0x2f,0x1f,0x99,0x0c,0x3e,
    0xda,0xb2,0xe8,0x5f,0x8b,0xe1,0x4b,0x88,0xed,0x3d,0xa8,0xda,0xe8,0x4d,0x45,
    0xb,0x8b,0x41,0xc8,0xac,0xf6,0x5a,0x75,0xe6,0xa5,0x76,0xe4,0xcb,0xa,
    0x33,0x8b,0x07,0xc9,0x73,0x0f,0x45,0x2a,0xc1,0xe3,0xe6,0xe7,0x21,0x66,
    0x1c,0x8b,0x8f,0xe6,0x3e,0x39,0x9a,0x2b,0xe2,0x8f,0xe0,0xe9c,0xb4,0x85,0x89,0x33,
    0x2a,0x0c,0xc8,0x5d,0x58,0xe1,0x89,0x12,0x8f,0x4d,0x2f,0x1f,0x99,0x0c,0x3e,
    0xda,0xb2,0xe8,0x5f,0x8b,0xe1,0x4b,0x88,0xed,0x3d,0xa8,0xda,0xe8,0x4d,0x45,
    0xb,0x8b,0x41,0xc8,0xac,0xf6,0x5a,0x75,0xe6,0xa5,0x76,0xe4,0xcb,0xa,
    0x33,0x8b,0x07,0xc9,0x73,0x0f,0x45,0x2a,0xc1,0xe3,0xe6,0xe7,0x21,0x66,
    0x1c,0x8b,0x8f,0xe6,0x3e,0x39,0x9a,0x2b,0xe2,0x8f,0xe0,0xe9c,0xb4,0x85,0x89,0x33,
    0x2a,0x0c,0xc8,0x5d,0x58,0xe1,0x89,0x12,0x8f,0x4d,0x2f,0x1f,0x99,0x0c,0x3e,
    0xda,0xb2,0xe8,0x5f,0x8b,0xe1,0x4b,0x88,0xed,0x3d,0xa8,0xda,0xe8,0x4d,0x45,
    0xb,0x8b,0x41,0xc8,0xac,0xf6,0x5a,0x75,0xe6,0xa5,0x76,0xe4,0xcb,0xa,
    0x33,0x8b,0x07,0xc9,0x73,0x0f,0x45,0x2a,0xc1,0xe3,0xe6,0xe7,0x21,0x66,
    0x1c,0x8b,0x8f,0xe6,0x3e,0x39,0x9a,0x2b,0xe2,0x8f,0xe0,0xe9c,0xb4,0x85,0x89,0x33,
    0x2a,0x0c,0xc8,0x5d,0x58,0xe1,0x89,0x12,0x8f,0x4d,0x2f,0x1f,0x99,0x0c,0x3e,
    0xda,0xb2,0xe8,0x5f,0x8b,0xe1,0x4b,0x88,0xed,0x3d,0xa8,0xda,0xe8,0x4d,0x45,
    0xb,0x8b,0x41,0xc8,0xac,0xf6,0x5a,0x75,0xe6,0xa5,0x76,0xe4,0xcb,0xa,
    0x33,0x8b,0x07,0xc9,0x73,0x0f,0x45,0x2a,0xc1,0xe3,0xe6,0xe7,0x21,0x66,
    0x1c,0x8b,0x8f,0xe6,0x3e,0x39,0x9a,0x2b,0xe2,0x8f,0xe0,0xe9c,0xb4,0x85,0x89,0x33,
    0x2a,0x0c,0xc8,0x5d,0x58,0xe1,0x89,0x12,0x8f,0x4d,0x2f,0x1f,0x99,0x0c,0x3e,
    0xda,0xb2,0xe8,0x5f,0x8b,0xe1,0x4b,0x88,0xed,0x3d,0xa8,0xda,0xe8,0x4d,0x45,
    0xb,0x8b,0x41,0xc8,0xac,0xf6,0x5a,0x75,0xe6,0xa5,0x76,0xe4,0xcb,0xa,
    0x33,0x8b,0x07,0xc9,0x73,0x0f,0x45,0x2a,0xc1,0xe3,0xe6,0xe7,0x21,0x66,
    0x1c,0x8b,0x8f,0xe6,0x3e,0x39,0x9a,0x2b,0xe2,0x8f,0xe0,0xe9c,0xb4,0x85,0x89,0x33,
    0x2a,0x0c,0xc8,0x5d,0x58,0xe1,0x89,0x12,0x8f,0x4d,0x2f,0x1f,0x99,0x0c,0x3e,
}};  #endif  // SHA512
10.1.10 SymmetricTestData.h

This is a vector for testing either encrypt or decrypt. The premise for decrypt is that the IV for decryption is
the same as the IV for encryption. However, the ivOut value may be different for encryption and
decryption. We will encrypt at least two blocks. This means that the chaining value will be used for each
of the schemes (if any) and that implicitly checks that the chaining value is handled properly.

```c
#if AES_128
const BYTE key_AES128 [] = {
  0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
  0xab, 0xf7, 0x15, 0xe8, 0x2f, 0x4f, 0x3c};
const BYTE dataIn_AES128 [] = {
  0x6b, 0xc1, 0xbe, 0xe2, 0xe4, 0x9f, 0x96,
  0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
  0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0xc9,
  0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51};
const BYTE dataOut_AES128_ECB [] = {
  0x3a, 0xd7, 0x7b, 0xb4, 0x0d, 0x7a, 0x36, 0x60,
  0xa8, 0x9e, 0xc9, 0xf3, 0x24, 0x66, 0xef, 0x9f,
  0xf5, 0xda, 0x5a, 0x96, 0xfd, 0xb9, 0x69, 0x9d,
  0xe7, 0x85, 0x89, 0x5a, 0x96, 0xfd, 0xb9, 0x69};
const BYTE dataOut_AES128_CBC [] = {
  0x76, 0x49, 0xab, 0xac, 0x81, 0x19, 0xb2, 0x46,
  0xce, 0xe9, 0x8e, 0x9b, 0x12, 0xe9, 0x19, 0x7d,
  0x50, 0x86, 0xcb, 0x9b, 0x50, 0x72, 0x19, 0x8e,
  0x95, 0x2d, 0x11, 0x3a, 0xb1, 0x91, 0x76, 0x7b};
const BYTE dataOut_AES128_CFB [] = {
  0x3b, 0x3f, 0xd9, 0x2e, 0xb7, 0x2d, 0xad, 0x20,
  0x33, 0x34, 0x49, 0xf8, 0xe8, 0x3c, 0xf9, 0x4a,
  0xc8, 0xa6, 0x45, 0x37, 0xa0, 0xb3, 0xa9, 0x3f,
  0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51};
const BYTE dataOut_AES128_OFB [] = {
  0x87, 0x4d, 0x61, 0x91, 0xb6, 0x20, 0xe3, 0x26,
  0x1b, 0xef, 0x68, 0x64, 0x99, 0x0d, 0xb6, 0xce,
  0x98, 0x06, 0xf6, 0x6b, 0x79, 0x70, 0xfd, 0xff,
  0x86, 0x17, 0x18, 0x7b, 0xb9, 0xff, 0xfd, 0xff};
#endif
#if AES_192
const BYTE key_AES192 [] = {
  0x8e, 0x73, 0xb0, 0xf7, 0xda, 0xe5, 0xa4, 0x4a,
  0xc8, 0x10, 0xf3, 0x2b, 0x80, 0x90, 0x79, 0xe5,
  0x62, 0xf8, 0xea, 0xd2, 0x52, 0x2c, 0x6b, 0x7b};
const BYTE dataIn_AES192 [] = {
  0x6b, 0xc1, 0xbe, 0xe2, 0xe4, 0x9f, 0x96,
  0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
  0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0xc9,
  0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51};
const BYTE dataOut_AES128_ECB [] = {
  0x3b, 0x3f, 0xd9, 0x2e, 0xb7, 0x2d, 0xad, 0x20,
  0x33, 0x34, 0x49, 0xf8, 0xe8, 0x3c, 0xf9, 0x4a,
  0xc8, 0xa6, 0x45, 0x37, 0xa0, 0xb3, 0xa9, 0x3f,
  0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51};
const BYTE dataOut_AES128_CBC [] = {
  0x76, 0x49, 0xab, 0xac, 0x81, 0x19, 0xb2, 0x46,
  0xce, 0xe9, 0x8e, 0x9b, 0x12, 0xe9, 0x19, 0x7d,
  0x50, 0x86, 0xcb, 0x9b, 0x50, 0x72, 0x19, 0x8e,
  0x95, 0x2d, 0x11, 0x3a, 0xb1, 0x91, 0x76, 0x7b};
const BYTE dataOut_AES128_CFB [] = {
  0x3b, 0x3f, 0xd9, 0x2e, 0xb7, 0x2d, 0xad, 0x20,
  0x33, 0x34, 0x49, 0xf8, 0xe8, 0x3c, 0xf9, 0x4a,
  0xc8, 0xa6, 0x45, 0x37, 0xa0, 0xb3, 0xa9, 0x3f,
  0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x51};
#endif
#endif
```
58 0x34, 0xc2, 0x59, 0x09, 0xc9, 0x9a, 0x41, 0x74,
59 0x67, 0xce, 0x7f, 0x7f, 0x81, 0x17, 0x36, 0x2f,
60 0x96, 0x1a, 0x2b, 0x70, 0x17, 0x1d, 0x3d, 0x7a);  
61 \textbf{const BYTE} dataOut\_AES192\_OFB [] = { 
62 0xcd, 0xc8, 0x0d, 0x6f, 0xdd, 0xf1, 0x8c, 0xab,
63 0x34, 0xc2, 0x59, 0x09, 0xc9, 0x9a, 0x41, 0x74,
64 0xfc, 0xc2, 0x8b, 0x8d, 0x4c, 0x63, 0x83, 0x7c,
65 0x09, 0xe8, 0x17, 0x00, 0xc1, 0x10, 0x04, 0x01}; 
66 \textbf{const BYTE} dataOut\_AES192\_CTR [] = { 
67 0x1a, 0xbc, 0x93, 0x24, 0x17, 0x52, 0x1c, 0xa2,
68 0x4f, 0x2b, 0x04, 0x59, 0xfe, 0x7e, 0x6e, 0x0b,
69 0x09, 0x03, 0x39, 0xec, 0x0a, 0xa6, 0xfa, 0xef,
70 0xd5, 0xc2, 0xc6, 0xf4, 0xce, 0x8e, 0x94}; 
71 \textbf{#endif}
72 \textbf{#if AES\_256}
73 \textbf{const BYTE} key\_AES256 [] = { 
74 0x60, 0x3d, 0xeb, 0x10, 0x15, 0xca, 0x71, 0xbe,
75 0x2b, 0x73, 0xae, 0xf0, 0x85, 0x7d, 0x77, 0x81,
76 0x1f, 0x35, 0x2c, 0x07, 0x3b, 0x61, 0x08, 0xd7,
77 0x2d, 0x98, 0x10, 0xa3, 0x09, 0x14, 0xdf, 0xf4}; 
78 \textbf{const BYTE} dataIn\_AES256 [] = { 
79 0x6b, 0xc1, 0xbe, 0xe2, 0x2e, 0x40, 0x9f, 0x96,
80 0xe9, 0x3d, 0x7e, 0x11, 0x73, 0x93, 0x17, 0x2a,
81 0xae, 0x2d, 0x8a, 0x57, 0x1e, 0x03, 0xac, 0x9c,
82 0x9e, 0xb7, 0x6f, 0xac, 0x45, 0xaf, 0x8e, 0x94}; 
83 \textbf{const BYTE} dataOut\_AES256\_ECB [] = { 
84 0xf3, 0xee, 0xd1, 0xbd, 0xb5, 0xd2, 0xa0, 0x3c,
85 0x06, 0x4b, 0x5a, 0x7e, 0x3d, 0xb1, 0x81, 0xf8,
86 0x59, 0x1c, 0xcb, 0x10, 0xd4, 0x10, 0xed, 0x26,
87 0xdc, 0x5b, 0xa7, 0x4a, 0x31, 0x36, 0x28, 0x70}; 
88 \textbf{const BYTE} dataOut\_AES256\_CBC [] = { 
89 0xf5, 0x8c, 0x4c, 0x04, 0xd6, 0xe5, 0xf1, 0xba,
90 0x77, 0x9e, 0xab, 0xfb, 0x5f, 0x7b, 0xfb, 0x6d,
91 0x9c, 0xfc, 0xe4, 0x96, 0x7e, 0xdb, 0x80, 0x8d,
92 0x67, 0x9f, 0x77, 0x7b, 0xc6, 0x70, 0x2c, 0x7d}; 
93 \textbf{const BYTE} dataOut\_AES256\_CFB [] = { 
94 0xdc, 0x7e, 0x84, 0xbf, 0xda, 0x79, 0x16, 0x4b,
95 0x7e, 0xc0, 0x84, 0x86, 0x98, 0x5d, 0x38, 0x60,
96 0x39, 0xff, 0xed, 0x14, 0x3b, 0x28, 0xb1, 0xc8,
97 0x32, 0x11, 0x3c, 0x63, 0x31, 0xe5, 0x40, 0x7b}; 
98 \textbf{const BYTE} dataOut\_AES256\_OFB [] = { 
99 0xdc, 0x7e, 0x84, 0xbf, 0xda, 0x79, 0x16, 0x4b,
100 0x7e, 0xc0, 0x84, 0x86, 0x98, 0x5d, 0x38, 0x60,
101 0x4f, 0xeb, 0xdc, 0x67, 0x40, 0xd2, 0x0b, 0x3a,
102 0xc8, 0x8f, 0x6a, 0xd8, 0x2a, 0x4f, 0xb0, 0x8d}; 
103 \textbf{const BYTE} dataOut\_AES256\_CTR [] = { 
104 0x60, 0x1e, 0xc3, 0x13, 0x77, 0x57, 0x89, 0xa5,
105 0xb7, 0xa7, 0xf5, 0x04, 0xbb, 0xf3, 0xd2, 0x28,
106 0xf4, 0x43, 0xe3, 0xca, 0x4d, 0x62, 0xb5, 0x9a,
107 0xca, 0x84, 0xe9, 0x90, 0xca, 0xca, 0xf5, 0xc5}; 
108 \textbf{#endif}
10.1.11 SymmetricTest.h

10.1.11.1 Introduction

This file contains the structures and data definitions for the symmetric tests. This file references the header file that contains the actual test vectors. This organization was chosen so that the program that is used to generate the test vector values does not have to also re-generate this data.

```c
#ifndef SELF_TEST_DATA
#error "This file many only be included in AlgorithmTests.c"
#endif

#ifndef _SYMMETRIC_TEST_H
#define _SYMMETRIC_TEST_H

#include "SymmetricTestData.h"

#endif // _SYMMETRIC_TEST_H
```

10.1.11.2 Symmetric Test Structures

```c
const SYMMETRIC_TEST_VECTOR c_symTestValues[NUM_SYMS] = {
    #ifndef COMMA
    #ifdef AES_128
    {ALG_AES_VALUE, 128, key_AES128, 16, sizeof(dataIn_AES128), dataIn_AES128,
     dataOut_AES128_CTR, dataOut_AES128_OFB, dataOut_AES128_CBC,
     dataOut_AES128_CFB, dataOut_AES128_ECB}
    # define COMMA,
    #endif
    #ifdef AES_192
    {ALG_AES_VALUE, 192, key_AES192, 16, sizeof(dataIn_AES192), dataIn_AES192,
     dataOut_AES192_CTR, dataOut_AES192_OFB, dataOut_AES192_CBC,
     dataOut_AES192_CFB, dataOut_AES192_ECB}
    # undef COMMA
    # define COMMA,
    #endif
    #ifdef AES_256
    {ALG_AES_VALUE, 256, key_AES256, 16, sizeof(dataIn_AES256), dataIn_AES256,
     dataOut_AES256_CTR, dataOut_AES256_OFB, dataOut_AES256_CBC,
     dataOut_AES256_CFB, dataOut_AES256_ECB}
    # undef COMMA
    # define COMMA,
    #endif
    #ifdef SM4_128
    {ALG_SM4_VALUE, 128, key_SM4128, 16, sizeof(dataIn_SM4128), dataIn_SM4128,
     dataOut_SM4128_CTR, dataOut_SM4128_OFB, dataOut_SM4128_CBC,
     dataOut_SM4128_CFB, dataOut_AES128_ECB}
    #endif
    #undef COMMA
    #endif
};
#endif // _SYMMETRIC_TEST_H
```
10.1.12 EccTestData.h

This file contains the parameter data for ECC testing.

```
#ifdef SELF_TEST_DATA

# Define the data for ECC Testing.

const TPM2B_EC_TEST c_ecTestKey_ds = {{32, {0xdf, 0x8d, 0xa4, 0xa3, 0x88, 0xf6, 0x76, 0x96, 0x89, 0xfc, 0x2f, 0x2d, 0xa1, 0xb4, 0x39, 0x7a, 0x78, 0xc4, 0x7f, 0x71, 0x8c, 0xa6, 0xa1, 0x85, 0xc0, 0xbf, 0xf3, 0x54, 0x20, 0x91, 0x2f, 0x73}}};

const TPM2B_EC_TEST c_ecTestKey_QsX = {{32, {0x17, 0xad, 0x2f, 0xc0, 0x7d, 0x2c, 0x42, 0x7f, 0x94, 0x01, 0x00, 0x35, 0x70, 0x80, 0x91, 0x85, 0xc0, 0xbf, 0xf3, 0x54, 0x20, 0x91, 0x2f, 0x73}}};

const TPM2B_EC_TEST c_ecTestKey_QsY = {{32, {0x22, 0x8d, 0xf5, 0x18, 0x45, 0x96, 0x6e, 0x58, 0x96, 0x89, 0xc0, 0xbf, 0xf3, 0x54, 0x20, 0x91, 0x2f, 0x73}}};

const TPM2B_EC_TEST c_ecTestKey_de = {{32, {0xb6, 0xb5, 0x33, 0x5c, 0xd1, 0xe2, 0x8f, 0x2b, 0x19, 0x18, 0x07, 0xc1, 0x8f, 0xda, 0x0c, 0x8d, 0x1f, 0xda, 0x9c, 0x12, 0x77, 0xc0, 0xf9}}};

const TPM2B_EC_TEST c_ecTestKey_QeX = {{32, {0xa5, 0x1e, 0x80, 0xd1, 0x76, 0x3e, 0x82, 0xbe, 0x5f, 0x33, 0x02, 0x25, 0x90, 0x3a, 0x90, 0x89, 0xe3, 0xe5, 0x10, 0x4a, 0xbc, 0x78, 0xa5, 0xc5, 0x07, 0x64, 0xaf, 0x91, 0xbc, 0xe6, 0xff, 0x85, 0x11, 0x40}}};

const TPM2B_EC_TEST c_ecTestEcdh_X = {{32, {0x64, 0x60, 0x92, 0x78, 0xdb, 0x33, 0x52, 0x07, 0x99, 0xe0, 0x8f, 0x8b, 0x19, 0x18, 0x07, 0xc1, 0x8f, 0xda, 0x0c, 0x8d, 0x1f, 0xda, 0x9c, 0x12, 0x77, 0xc0, 0xf9}}};

const TPM2B_EC_TEST c_ecTestEcdh_Y = {{32, {0x58, 0x94, 0x05, 0x82, 0xbe, 0x5f, 0x33, 0x02, 0x25, 0x90, 0x3a, 0x90, 0x89, 0xe3, 0xe5, 0x10, 0x4a, 0xbc, 0x78, 0xa5, 0xc5, 0x07, 0x64, 0xaf, 0x91, 0xbc, 0xe6, 0xff, 0x85, 0x11, 0x40}}};

const TPM2B_EC_TEST c_ecTestValue = {{64, {0x78, 0xd5, 0xd4, 0x56, 0x43, 0x61, 0xdb, 0x97, 0xa4, 0x32, 0xc4, 0x0b, 0x06, 0xa9, 0xa8, 0xa0, 0x1c, 0x65, 0xcc, 0xa7, 0xc9, 0x19, 0x10, 0x06, 0x54, 0x1b, 0x90, 0xe7, 0x7d, 0xa1, 0x77, 0x94, 0x65, 0x1c, 0x3e, 0xc1, 0x3e}}};

```

The static key

```
const TPM2B_EC_TEST c_ecTestKey_ds = {{32, {0xdf, 0x8d, 0xa4, 0xa3, 0x88, 0xf6, 0x76, 0x96, 0x89, 0xfc, 0x2f, 0x2d, 0xa1, 0xb4, 0x39, 0x7a, 0x78, 0xc4, 0x7f, 0x71, 0x8c, 0xa6, 0xa1, 0x85, 0xc0, 0xbf, 0xf3, 0x54, 0x20, 0x91, 0x2f, 0x73}}};
```

The ephemeral key

```
```

ECDH test results

```
const TPM2B_EC_TEST c_ecTestEcdh_X = {{32, {0x64, 0x60, 0x92, 0x78, 0xdb, 0x33, 0x52, 0x07, 0x99, 0xe0, 0x8f, 0x8b, 0x19, 0x18, 0x07, 0xc1, 0x8f, 0xda, 0x0c, 0x8d, 0x1f, 0xda, 0x9c, 0x12, 0x77, 0xc0, 0xf9}}};
```

```
```
0xf8,0xf2,0xf4,0xa5,0xe0,0x1d,0x3c,0xa2,0x39,0x31,0xe4,0xe7,0x36,0x3b,0xb5,0x5f}}};
const TPM2B_EC_TEST c_TestEcDsa_s = {{32, {
0x8f,0xd0,0x12,0xd9,0x24,0x75,0xf6,0xc4,0x3b,0xb5,0x46,0x75,0x95,0xa5,0x39,0x0a,0x14,0x49,0x3d,0x2f}}};
const TPM2B_EC_TEST c_TestEcSchnorr_r = {{32, {
0xf7,0xb9,0x15,0x4c,0x34,0xf6,0x41,0xa3,0xd2,0xf1,0xd2,0xf4,0x13,0x6a,0x4f,
0x63,0xb8,0x4d,0xb5,0xc8,0xcd,0xde,0x85,0x95,0xa5,0x39,0x0a,0x14,0x49,0x3d,0x2f}}};
const TPM2B_EC_TEST c_TestEcSchnorr_s = {{32,{
0xfe,0xbe,0x17,0xaa,0x31,0x22,0x9f,0xd0,0xd2,0xf5,0x25,0x04,0x92,0xb0,0xaa,0xe4,
0x0c,0x1c,0x0b,0x79,0x6d,0x42,0xb3,0x4e,0x3f,0xbb,0xe2,0x5f,0xd0,0x8b,0x3c}}};
#endif // SHA256
#elif ALG_SHA384_VALUE == DEFAULT_TEST_HASH
const TPM2B_EC_TEST c_TestEcDsa_r = {{32, {
0xf5,0x74,0x6d,0xd6,0xc6,0x56,0x86,0xbb,0xba,0x1c,0xba,0x75,0x75,0x65,0x6e,0x64,0x31,
0x5b,0x0e,0x3,0x9f,0x24,0x3f,0xb0,0xe0,0x1b,0xe0,0x7e,0xe0,0xad,0xcb,0x82}}};
const TPM2B_EC_TEST c_TestEcDsa_s = {{32,{
0xc2,0x4f,0x32,0xa1,0x06,0xc0,0x85,0x4e,0xc6,0xd0,0x31,0x91,0x9f,0x79,0xc6,
0x5b,0xe5,0x7b,0x94,0xa1,0x91,0x38,0xc4,0x20,0xa2,0x10,0xf0,0xd5,0x9d,0x9f}}};
const TPM2B_EC_TEST c_TestEcSchnorr_r = {{32, {
0x1e,0xb8,0xe1,0xc0,0x9e,0x39,0x1e,0xb9,0xa2,0xe6,0x59,0xda,0x6a,0x30,
0x6a,0x1f,0x1c,0x4f,0x36,0x19,0xc1,0xe3,0x30,0x4a,0x85,0x1b,0xe9,0x74,0x35,0x66}}};
const TPM2B_EC_TEST c_TestEcSchnorr_s = {{32, {
0xb9,0xe6,0x3e,0x7e,0xcb,0xb9,0xe2,0xf1,0xcc,0xf4,0x48,0x4a,0xda,0xc8,0x87,
0x87,0xb4,0xba,0x40,0xe2,0x5b,0x68,0x11,0x14,0xcf,0xa0,0x0e,0x85,0x46,0x99,0x01}}};
#endif // SHA384
#elif ALG_SHA512_VALUE == DEFAULT_TEST_HASH
const TPM2B_EC_TEST c_TestEcDsa_r = {{32, {
0xc9,0x71,0xe6,0xb4,0xe2,0x0f,0x46,0x26,0x8c,0x27,0x00,0x06,0x3b,0x00,0x0f,0xa3,0x17,
0x72,0x48,0x40,0x49,0x4d,0x51,0x4f,0xa4,0xc6,0xe7,0xe6,0xe9,0xe7,0xb4,0x79,0xb2}}};
const TPM2B_EC_TEST c_TestEcDsa_s = {{32,{
0x87,0xbc,0xe0,0xed,0x74,0x60,0x9e,0xfa,0xe8,0x16,0xf3,0xf9,0xe6,0x26,0x07,
0x3c,0x74,0x31,0x7e,0xf0,0x62,0x46,0xdc,0xe6,0x45,0x22,0x47,0x3e,0xe0,0x0a,0x02}}};
const TPM2B_EC_TEST c_TestEcSchnorr_r = {{32, {
0xcc,0x07,0xad,0x65,0x91,0xdd,0xa0,0x10,0x23,0xae,0x53,0xec,0xdf,0xf1,0x50,0x90,
0x16,0x96,0xf4,0x45,0x09,0x73,0xc9,0x84,0xb5,0x5c,0x5f,0x08,0x51,0xcb,0x60,0x01}}};
const TPM2B_EC_TEST c_TestEcSchnorr_s = {{32, {
0x55,0x20,0x21,0x54,0xe2,0x49,0x07,0x47,0x71,0xf4,0x99,0x15,0x54,0xf3,0xbab,0x14,
0xdb,0x8e,0xda,0x79,0xb6,0x02,0xe0,0xe3,0x5e,0x6f,0x2c,0xb6,0x05,0xbd,0x14,0x10}}};
#endif // SHA512
#endif // SELF_TEST_DATA
10.1.13 CryptSym.h

10.1.13.1 Introduction

This file contains the implementation of the symmetric block cipher modes allowed for a TPM. These functions only use the single block encryption functions of the selected symmetric cryptographic library.

10.1.13.2 Includes, Defines, and Typedefs

```c
#ifndef CRYPT_SYM_H
#define CRYPT_SYM_H

typedef union tpmCryptKeySchedule_t {
  #if ALG_AES
  tpmKeyScheduleAES AES;
  #endif
  #if ALG_SM4
  tpmKeyScheduleSM4 SM4;
  #endif
  #if ALG_CAMELLIA
  tpmKeyScheduleCAMELLIA CAMELLIA;
  #endif
  #if ALG_TDES
  tpmKeyScheduleTDES TDES[3];
  #endif
  #if SYMMETRIC_ALIGNMENT == 8
  uint64_t alignment;
  #else
  uint32_t alignment;
  #endif
} tpmCryptKeySchedule_t;
```

Each block cipher within a library is expected to conform to the same calling conventions with three parameters (keySchedule, in, and out) in the same order. That means that all algorithms would use the same order of the same parameters. The code is written assuming the (keySchedule, in, and out) order. However, if the library uses a different order, the order can be changed with a SWIZZLE macro that puts the parameters in the correct order. Note that all algorithms have to use the same order and number of parameters because the code to build the calling list is common for each call to encrypt or decrypt with the algorithm chosen by setting a function pointer to select the algorithm that is used.

```c
#define ENCRYPT(keySchedule, in, out) \
  encrypt(SWIZZLE(keySchedule, in, out))
#define DECRYPT(keySchedule, in, out) \
  decrypt(SWIZZLE(keySchedule, in, out))
```

Note that the macros rely on encrypt as local values in the functions that use these macros. Those parameters are set by the macro that set the key schedule to be used for the call.

```c
#define ENCRYPT_CASE(ALG) \ 
  case TPM_ALG_##ALG: \ 
    TpmCryptSetEncryptKey##ALG(key, keySizeInBits, &keySchedule.ALG); \ 
    encrypt = (TpmCryptSetSymKeyCall_t)TpmCryptEncrypt##ALG; \ 
    break;
#define DECRYPT_CASE(ALG) \ 
  case TPM_ALG_##ALG: \ 
    TpmCryptSetDecryptKey##ALG(key, keySizeInBits, &keySchedule.ALG); \ 
    decrypt = (TpmCryptSetSymKeyCall_t)TpmCryptDecrypt##ALG; \ 
    break;
#if ALG_AES
#define ENCRYPT_CASE_AES ENCRYPT_CASE(AES)
```
#define DECRYPT_CASE_AES DECRYPT_CASE(AES)
#else
#define ENCRYPT_CASE_AES
#define DECRYPT_CASE_AES
#endif
#if ALG_SM4
#define ENCRYPT_CASE_SM4 ENCRYPT_CASE(SM4)
#define DECRYPT_CASE_SM4
#else
#define ENCRYPT_CASE_SM4
#define DECRYPT_CASE_SM4
#endif
#if ALG_CAMELLIA
#define ENCRYPT_CASE_CAMELLIA ENCRYPT_CASE(CAMELLIA)
#define DECRYPT_CASE_CAMELLIA
#else
#define ENCRYPT_CASE_CAMELLIA
#define DECRYPT_CASE_CAMELLIA
#endif
#if ALG_TDES
#define ENCRYPT_CASE_TDES ENCRYPT_CASE(TDES)
#define DECRYPT_CASE_TDES
#else
#define ENCRYPT_CASE_TDES
#define DECRYPT_CASE_TDES
#endif

For each algorithm the case will either be defined or null.

#define SELECT(direction)  
  switch(algorithm) {  
    case AES:  
    case SM4:  
    case CAMELLIA:  
    case TDES:  
      default:  
      FAIL(FATAL_ERROR_INTERNAL);
    }
#endif // CRYPT_SYM_H
10.2  Source

10.2.1  AlgorithmTests.c

10.2.1.1  Introduction

This file contains the code to perform the various self-test functions.

NOTE: In this implementation, large local variables are made static to minimize stack usage, which is critical for stack-constrained platforms.

10.2.1.2  Includes and Defines

```c
#include "Tpm.h"
define SELF_TEST_DATA
#if SELF_TEST

These includes pull in the data structures. They contain data definitions for the various tests.

#include "SelfTest.h"
#include "SymmetricTest.h"
#include "RsaTestData.h"
#include "EccTestData.h"
#include "HashTestData.h"
#include "KdfTestData.h"
define TEST_DEFAULT_TEST_HASH(vector)
  if (TEST_BIT(DEFAULT_TEST_HASH, g_toTest))
    TestHash(DEFAULT_TEST_HASH, vector);

Make sure that the algorithm has been tested

define CLEAR_BOTH(alg) { CLEAR_BIT(alg, *toTest); }
define SET_BOTH(alg) { SET_BIT(alg, *toTest); }
define TEST_BOTH(alg) { (toTest != &g_toTest) ? TEST_BIT(alg, *toTest) || TEST_BIT(alg, g_toTest) : TEST_BIT(alg, *toTest) }

can only cancel if doing a list.

define CHECK_CANCELED
  if (_plat__IsCanceled() && toTest != &g_toTest)
    return TPM_RC_CANCELED;
```

10.2.1.3  Hash Tests

10.2.1.3.1  Description

The hash test does a known-value HMAC using the specified hash algorithm.

10.2.1.3.2  TestHash()

The hash test function.
25  static TPM_RC
26  TestHash(
27      TPM_ALG_ID          hashAlg,
28      ALGORITHM_VECTOR  *toTest
29  )
30  {
31      static TPM2B_DIGEST    computed; // value computed
32      static HMAC_STATE    state;
33      UINT16                   digestSize;
34      const TPM2B           *testDigest = NULL;
35      // TPM2B_TYPE(HMAC_BLOCK, DEFAULT_TEST_HASH_BLOCK_SIZE);
36
37      pAssert(hashAlg != ALG_NULL_VALUE);
38      switch(hashAlg)
39      {
40        #if ALG_SHA1
41          case ALG_SHA1_VALUE:
42              testDigest = &c_SHA1_digest.b;
43              break;
44        #endif
45        #if ALG_SHA256
46          case ALG_SHA256_VALUE:
47              testDigest = &c_SHA256_digest.b;
48              break;
49        #endif
50        #if ALG_SHA384
51          case ALG_SHA384_VALUE:
52              testDigest = &c_SHA384_digest.b;
53              break;
54        #endif
55        #if ALG_SHA512
56          case ALG_SHA512_VALUE:
57              testDigest = &c_SHA512_digest.b;
58              break;
59        #endif
60        #if ALG_SM3_256
61          case ALG_SM3_256_VALUE:
62              testDigest = &c_SM3_256_digest.b;
63              break;
64        #endif
65        default:
66            FAIL(FATAL_ERROR_INTERNAL);
67      }
68      // Clear the to-test bits
69      CLEAR_BOTH(hashAlg);
70      // Set the HMAC key to twice the digest size
71      digestSize = CryptHashGetDigestSize(hashAlg);
72      CryptHmacStart(&state, hashAlg, digestSize * 2,
73                      (BYTE *)c_hashTestKey.t.buffer);
74      CryptDigestUpdate(&state.hashState, 2 * CryptHashGetBlockSize(hashAlg),
75                          (BYTE *)c_hashTestData.t.buffer);
76      computed.t.size = digestSize;
77      CryptHmacEnd(&state, digestSize, computed.t.buffer);
78      if((testDigest->size != computed.t.size)
79          || (memcmp(testDigest->buffer, computed.t.buffer, computed.b.size) != 0))
80          SELF_TEST_FAILURE;
81      return TPM_RC_SUCCESS;
82  }
10.2.1.4 Symmetric Test Functions

10.2.1.4.1 MakeIv()

Internal function to make the appropriate IV depending on the mode.

```c
static UINT32
MakeIv(  
TPM_ALG_ID    mode, // IN: symmetric mode  
BYTE         *iv // OUT: IV to fill in  
)
{
    BYTE i;
    if(mode == ALG_ECB_VALUE)
        return 0;
    if(mode == ALG_CTR_VALUE)
    {
        // The test uses an IV that has 0xff in the last byte
        for(i = 1; i <= size; i++)
            *iv++ = 0xff - (BYTE)(size - i);
    }
    else
    {
        for(i = 0; i < size; i++)
            *iv++ = i;
    }
    return size;
}
```

10.2.1.4.2 TestSymmetricAlgorithm()

Function to test a specific algorithm, key size, and mode.

```c
static void
TestSymmetricAlgorithm(  
    const SYMMETRIC_TEST_VECTOR     *test, //  
    TPM_ALG_ID                       mode //  
)
{
    static BYTE                 encrypted[MAX_SYM_BLOCK_SIZE * 2];
    static BYTE                 decrypted[MAX_SYM_BLOCK_SIZE * 2];
    static TPM2B_IV             iv;
    // Get the appropriate IV
    iv.t.size = (UINT16)MakeIv(mode, test->ivSize, iv.t.buffer);
    // Encrypt known data
    CryptSymmetricEncrypt(encrypted, test->alg, test->keyBits, test->key, &iv,
        mode, test->dataInOutSize, test->dataIn);
    // Check that it matches the expected value
    if(!MemoryEqual(encrypted, test->dataOut[mode - ALG_CTR_VALUE],
        test->dataInOutSize))
        SELF_TEST_FAILURE;
    // Reinitialize the iv for decryption
    MakeIv(mode, test->ivSize, iv.t.buffer);
    CryptSymmetricDecrypt(decrypted, test->alg, test->keyBits, test->key, &iv,
        mode, test->dataInOutSize,
        test->dataOut[mode - ALG_CTR_VALUE]);
    // Make sure that it matches what we started with
    if(!MemoryEqual(decrypted, test->dataIn, test->dataInOutSize))
        SELF_TEST_FAILURE;
```
10.2.1.4.3 AllSymsAreDone()

Checks if both symmetric algorithms have been tested. This is put here so that addition of a symmetric algorithm will be relatively easy to handle

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>all symmetric algorithms tested</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>not all symmetric algorithms tested</td>
</tr>
</tbody>
</table>

```c
static BOOL AllSymsAreDone(
    ALGORITHM_VECTOR        *toTest
) {
    return (!TEST_BOTH(ALG_AES_VALUE) && !TEST_BOTH(ALG_SM4_VALUE));
}
```

10.2.1.4.4 AllModesAreDone()

Checks if all the modes have been tested

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>all modes tested</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>all modes not tested</td>
</tr>
</tbody>
</table>

```c
static BOOL AllModesAreDone(
    ALGORITHM_VECTOR            *toTest
) {
    TPM_ALG_ID                  alg;
    for(alg = TPM_SYM_MODE_FIRST; alg <= TPM_SYM_MODE_LAST; alg++)
        if(TEST_BOTH(alg))
            return FALSE;
    return TRUE;
}
```

10.2.1.4.5 TestSymmetric()

If alg is a symmetric block cipher, then all of the modes that are selected are tested. If alg is a mode, then all algorithms of that mode are tested.

```c
static TPM_RC TestSymmetric(
    TPM_ALG_ID                   alg,
    ALGORITHM_VECTOR            *toTest
) {
    SYM_INDEX                    index;
    TPM_ALG_ID                   mode;
    // if(!TEST_BIT(alg, *toTest))
        return TPM_RC_SUCCESS;
    if(alg == ALG_AES_VALUE || alg == ALG_SM4_VALUE || alg == ALG_CAMELLIA_VALUE)
        
```
// Will test the algorithm for all modes and key sizes
CLEAR_BOTH(alg);

// A test this algorithm for all modes
for(index = 0; index < NUM_SYMS; index++)
{
    if(c_symTestValues[index].alg == alg)
    {
        for(mode = TPM_SYM_MODE_FIRST;
            mode <= TPM_SYM_MODE_LAST;
            mode++)
        {
            if(TEST_BIT(mode, *toTest))
                TestSymmetricAlgorithm(&c_symTestValues[index], mode);
        }
    }

    // if all the symmetric tests are done
    if(AllSymsAreDone(toTest))
    {
        // all symmetric algorithms tested so no modes should be set
        for(alg = TPM_SYM_MODE_FIRST; alg <= TPM_SYM_MODE_LAST; alg++)
            CLEAR_BOTH(alg);
    }

    else if(TPM_SYM_MODE_FIRST <= alg && alg <= TPM_SYM_MODE_LAST)
    {
        // Test this mode for all key sizes and algorithms
        for(index = 0; index < NUM_SYMS; index++)
        {
            // The mode testing only comes into play when doing self tests
            // by command. When doing self tests by command, the block ciphers are
            // tested first. That means that all of their modes would have been
            // tested for all key sizes. If there is no block cipher left to
            // test, then clear this mode bit.
            if(!TEST_BIT(ALG_AES_VALUE, *toTest)
                && !TEST_BIT(ALG_SM4_VALUE, *toTest))
            {
                CLEAR_BOTH(alg);
            }

            else
            {
                for(index = 0; index < NUM_SYMS; index++)
                {
                    if(TEST_BIT(c_symTestValues[index].alg, *toTest))
                        TestSymmetricAlgorithm(&c_symTestValues[index], alg);
                }

                // have tested this mode for all algorithms
                CLEAR_BOTH(alg);
            }
        }

        // if all the modes are done
        if(AllModesAreDone(toTest))
        {
            CLEAR_BOTH(ALG_AES_VALUE);
            CLEAR_BOTH(ALG_SM4_VALUE);
        }

        else
            pAssert(alg == 0 && alg != 0);
    }
}

10.2.1.5 RSA Tests

#if ALG_RSA
10.2.1.5.1 Introduction

The tests are for public key only operations and for private key operations. Signature verification and encryption are public key operations. They are tested by using a KVT. For signature verification, this means that a known good signature is checked by CryptRsaValidateSignature(). If it fails, then the TPM enters failure mode. For encryption, the TPM encrypts known values using the selected scheme and checks that the returned value matches the expected value.

For private key operations, a full scheme check is used. For a signing key, a known key is used to sign a known message. Then that signature is verified. Since the signature may involve use of random values, the signature will be different each time and we can’t always check that the signature matches a known value. The same technique is used for decryption (RSADP/RSAEP).

When an operation uses the public key and the verification has not been tested, the TPM will do a KVT.

The test for the signing algorithm is built into the call for the algorithm.

10.2.1.5.2 RsaKeyInitialize()

The test key is defined by a public modulus and a private prime. The TPM’s RSA code computes the second prime and the private exponent.

static void RsaKeyInitialize(
  OBJECT *testObject
)
{
  MemoryCopy2B(&testObject->publicArea.unique.rsa.b, (P2B)&c_rsaPublicModulus,
  sizeof(c_rsaPublicModulus));
  MemoryCopy2B(&testObject->sensitive.sensitive.rsa.b, (P2B)&c_rsaPrivatePrime,
  sizeof(testObject->sensitive.sensitive.rsa.t.buffer));
  testObject->publicArea.parameters.rsaDetail.keyBits = RSA_TEST_KEY_SIZE * 8;
  // Use the default exponent
  testObject->publicArea.parameters.rsaDetail.exponent = 0;
  testObject->attributes.privateExp = 0;
}

10.2.1.5.3 TestRsaEncryptDecrypt()

These test are for an public key encryption that uses a random value.

static TPM_RC TestRsaEncryptDecrypt(
  TPM_ALG_ID scheme, // IN: the scheme
  ALGORITHM_VECTOR *toTest //
)
{
  static TPM2B_PUBLIC_KEY_RSA testInput;
  static TPM2B_PUBLIC_KEY_RSA testOutput;
  static OBJECT testObject;
  const TPM2B_RSA_TEST_KEY *kvtValue = NULL;
  TPM_RC result = TPM_RC_SUCCESS;
  const TPM2B *testLabel = NULL;
  TPM_T椽RSA_DECRYPT rsaScheme;
  //
  // Don’t need to initialize much of the test object but do need to initialize
  // the flag indicating that the private exponent has been computed.
  testObject.attributes.privateExp = CLEAR;
  RsaKeyInitialize(&testObject);
  rsaScheme.scheme = scheme;
  rsaScheme.details.anySig.hashAlg = DEFAULT_TEST_HASH;
  CLEAR_BOTH(scheme);
CLEAR_BOTH(ALG_NULL_VALUE);

if(scheme == ALG_NULL_VALUE)
{
    // This is an encryption scheme using the private key without any encoding.
    memcpy(testInput.t.buffer, c_RsaTestValue, sizeof(c_RsaTestValue));
    testInput.t.size = sizeof(c_RsaTestValue);
    if(TPM_RC_SUCCESS != CryptRsaEncrypt(&testOutput, &testInput.b,
        &testObject, &rsaScheme, NULL, NULL))
        SELF_TEST_FAILURE;
    if(!MemoryEqual(testOutput.t.buffer, c_RsaepKvt.buffer, c_RsaepKvt.size))
        SELF_TEST_FAILURE;
    MemoryCopy2B(&testInput.b, &testOutput.b, sizeof(testInput.t.buffer));
    if(TPM_RC_SUCCESS != CryptRsaDecrypt(&testOutput.b, &testInput.b,
        &testObject, &rsaScheme, NULL))
        SELF_TEST_FAILURE;
    if(!MemoryEqual(testOutput.t.buffer, c_RsaTestValue,
        sizeof(c_RsaTestValue)))
        SELF_TEST_FAILURE;
}
else
{
    // ALG_RSAES_VALUE:
    // This is a decryption scheme using padding according to
    // PKCS#1v2.1, 7.2. This padding uses random bits. To test a public
    // key encryption that uses random data, encrypt a value and then
    // decrypt the value and see that we get the encrypted data back.
    // The hash is not used by this encryption so it can be TMP_ALG_NULL
    // ALG_OAEP_VALUE:
    // This is also an decryption scheme and it also uses a
    // pseudo-random
    // value. However, this also uses a hash algorithm. So, we may need
    // to test that algorithm before use.
    if(scheme == ALG_OAEP_VALUE)
    {
        TEST_DEFAULT_TEST_HASH(toTest);
        kvtValue = &c_OaepKvt;
        testLabel = OAEP_TEST_STRING;
    }
    else if(scheme == ALG_RSAES_VALUE)
    {
        kvtValue = &c_RsaesKvt;
        testLabel = NULL;
    }
    else
        SELF_TEST_FAILURE;

    // Only use a digest-size portion of the test value
    memcpy(testInput.t.buffer, c_RsaTestValue, DEFAULT_TEST_DIGEST_SIZE);
    testInput.t.size = DEFAULT_TEST_DIGEST_SIZE;

    // See if the encryption works
    if(TPM_RC_SUCCESS != CryptRsaEncrypt(&testOutput, &testInput.b,
        &testObject, &rsaScheme, testLabel, NULL))
        SELF_TEST_FAILURE;
    MemoryCopy2B(&testInput.b, &testOutput.b, sizeof(testInput.t.buffer));
    if(TPM_RC_SUCCESS != CryptRsaDecrypt(&testOutput.b, &testInput.b,
        &testObject, &rsaScheme, testLabel))
        SELF_TEST_FAILURE;
    // See if the results compare
    if(testOutput.t.size != DEFAULT_TEST_DIGEST_SIZE
        || !MemoryEqual(testOutput.t.buffer, c_RsaTestValue,
            DEFAULT_TEST_DIGEST_SIZE))
        SELF_TEST_FAILURE;

    // Now check that the decryption works on a known value
}
MemoryCopy2B(&testInput.b, (P2B)kvtValue, sizeof(testInput.t.buffer));
if(TPM_RC_SUCCESS != CryptRsaDecrypt(&testOutput.b, &testInput.b, &testObject, &rsaScheme, testLabel))
    SELF_TEST_FAILURE;
if(testOutput.t.size != DEFAULT_TEST_DIGEST_SIZE
    || !MemoryEqual(testOutput.t.buffer, c_RsaTestValue, DEFAULT_TEST_DIGEST_SIZE))
    SELF_TEST_FAILURE;
} return result;
}

10.2.1.5.4 TestRsaSignAndVerify()

This function does the testing of the RSA sign and verification functions. This test does a KVT.

static TPM_RC
TestRsaSignAndVerify(
    TPM_ALG_ID               scheme,
    ALGORITHM_VECTOR        *toTest
)
{
    TPM_RC                      result = TPM_RC_SUCCESS;
    static OBJECT               testObject;
    static TPM2B_DIGEST         testDigest;
    static TPMT_SIGNATURE       testSig;

    // Do a sign and signature verification.
    // RSASSA:
    // This is a signing scheme according to PKCS#1-v2.1 8.2. It does not
    // use random data so there is a KVT for the signing operation. On
    // first use of the scheme for signing, use the TPM's RSA key to
    // sign a portion of c_RsaTestData and compare the results to c_RsassaKvt. Then
    // decrypt the data to see that it matches the starting value. This verifies
    // the signature with a KVT
    // Clear the bits indicating that the function has not been checked. This is to
    // prevent looping
    CLEAR_BOTH(scheme);
    CLEAR_BOTH(ALG_NULL_VALUE);
    CLEAR_BOTH(ALG_RSA_VALUE);

    RsaKeyInitialize(&testObject);
    memcpy(testDigest.t.buffer, (BYTE *)c_RsaTestValue, DEFAULT_TEST_DIGEST_SIZE);
    testDigest.t.size = DEFAULT_TEST_DIGEST_SIZE;
    testSig.sigAlg = scheme;
    testSig.signature.rsapss.hash = DEFAULT_TEST_HASH;

    // RSAPSS:
    // This is a signing scheme a according to PKCS#1-v2.2 8.1 it uses
    // random data in the signature so there is no KVT for the signing
    // operation. To test signing, the TPM will use the TPM's RSA key
    // to sign a portion of c_RsaTestValue and then it will verify the
    // signature. For verification, c_RsapssKvt is verified before the
    // user signature blob is verified. The worst case for testing of this
    // algorithm is two private and one public key operation.
    // The process is to sign known data. If RSASSA is being done, verify that the
    // signature matches the precomputed value. For both, use the signed value and
    // see that the verification says that it is a good signature. Then
    // if testing RSAPSS, do a verify of a known good signature. This ensures that
    // the validation function works.
if (TPM_RC_SUCCESS != CryptRsaSign(&testSig, &testObject, &testDigest, NULL))
    SELF_TEST_FAILURE;

// For RSASSA, make sure the results is what we are looking for
if (testSig.sigAlg == ALG_RSASSA_VALUE)
    {
        if (testSig.signature.rsassa.sig.t.size != RSA_TEST_KEY_SIZE
            || !MemoryEqual(c_RsassaKvt.buffer, 
                           testSig.signature.rsassa.sig.t.buffer, 
                           RSA_TEST_KEY_SIZE))
            SELF_TEST_FAILURE;
    }

// See if the TPM will validate its own signatures
if (TPM_RC_SUCCESS != CryptRsaValidateSignature(&testSig, &testObject, 
                                                  &testDigest))
    SELF_TEST_FAILURE;

// If this is RSAPSS, check the verification with known signature
// Have to copy because CryptRsaValidateSignature() eats the signature
if (ALG_RSAPSS_VALUE == scheme)
    {
        MemoryCopy2B(&testSig.signature.rsapss.sig.b, (P2B)&c_RsapssKvt, 
                     sizeof(testSig.signature.rsapss.sig.t.buffer));
        if (TPM_RC_SUCCESS != CryptRsaValidateSignature(&testSig, &testObject, 
                                                         &testDigest))
            SELF_TEST_FAILURE;
    }

return result;

10.2.1.5.5 TestRSA()

Function uses the provided vector to indicate which tests to run. It will clear the vector after each test is run and also clear g_toTest

static TPM_RC TestRsa(TPM_ALG_ID alg, 
                      ALGORITHM_VECTOR *toTest)
    {
        TPM_RC result = TPM_RC_SUCCESS;

        // switch(alg)
        {
            case ALG_NULL_VALUE:
                // This is the RSAEP/RSADP function. If we are processing a list, don't
                // need to test these now because any other test will validate
                // RSAEP/RSADP. Can tell this is list of test by checking to see if
                // 'toTest' is pointing at g_toTest. If so, this is an isolated test
                // an need to go ahead and do the test;
                if ((toTest == &g_toTest)
                    || (!TEST_BIT(ALG_RSASSA_VALUE, *toTest)
                        && !TEST_BIT(ALG_RSAES_VALUE, *toTest)
                        && !TEST_BIT(ALG_RSAPSS_VALUE, *toTest)
                        && !TEST_BIT(ALG_OAEP_VALUE, *toTest)))
                    // Not running a list of tests or no other tests on the list
                    // so run the test now
                    result = TestRsaEncryptDecrypt(alg, toTest);
                // if not running the test now, leave the bit on, just in case things
                // get interrupted
                break;
            case ALG_OAEP_VALUE:
            case ALG_RSAES_VALUE:
                result = TestRsaEncryptDecrypt(alg, toTest);
                break;
10.2.1.6 ECC Tests

10.2.1.6.1 LoadEccParameter()

This function is mostly for readability and type checking

```c
static void
LoadEccParameter(
    TPM2B_ECC_PARAMETER *to,      // target
    const TPM2B_EC_TEST *from,     // source
)
{
    MemoryCopy2B(&to->b, &from->b, sizeof(to->t.buffer));
}
```

10.2.1.6.2 LoadEccPoint()

```c
static void
LoadEccPoint(
    TPMS_ECC_POINT *point,         // target
    const TPM2B_EC_TEST *x,        // source
    const TPM2B_EC_TEST *y
)
{
    MemoryCopy2B(&point->x.b, (TPM2B *)x, sizeof(point->x.t.buffer));
    MemoryCopy2B(&point->y.b, (TPM2B *)y, sizeof(point->y.t.buffer));
}
```

10.2.1.6.3 TestECDH()

This test does a KVT on a point multiply.

```c
static TPM_RC
TestECDH(
    TPM_ALG_ID scheme,           // IN: for consistency
    ALGORITHM_VECTOR *toTest     // IN/OUT: modified after test is run
)
{
    static TPMS_ECC_POINT Z;
    static TPMS_ECC_POINT Qe;
    static TPM2B_ECC_PARAMETER ds;
    TPM_RC result = TPM_RC_SUCCESS;

    if(TPM_RC_SUCCESS != CryptEccPointMultiply(&Z, c_testCurve, &Qe, &ds, &c_ecTestKey_QeX, &c_ecTestKey_QeY))
        return TPM_RC_FAILURE;
}
```
SELF_TEST_FAILURE;

if (!MemoryEqual2B(&c_ecTestEcdh_X.b, &Z.x.b)
   || !MemoryEqual2B(&c_ecTestEcdh_Y.b, &Z.y.b))
   SELF_TEST_FAILURE;
return result;
}

10.2.1.6.4 TestEccSignAndVerify()

static TPM_RC TestEccSignAndVerify(
    TPM_ALG_ID scheme,
    ALGORITHM_VECTOR *toTest)
{
    static OBJECT testObject;
    static TPMT_SIGNATURE testSig;
    static TPMT_ECC_SCHEME eccScheme;

    testSig.sigAlg = scheme;
testSig.signature.ecdsa.hash = DEFAULT_TEST_HASH;
    eccScheme.scheme = scheme;
    eccScheme.details.anySig.hashAlg = DEFAULT_TEST_HASH;
    CLEAR_BOTH(scheme);
    CLEAR_BOTH(ALG_ECDH_VALUE);

    // ECC signature verification testing uses a KVT.
    switch(scheme)
    {
    case ALG_ECDSA_VALUE:
        LoadEccParameter(&testSig.signature.ecdsa.signatureR, &c_TestEcDsa_r);
        LoadEccParameter(&testSig.signature.ecdsa.signatureS, &c_TestEcDsa_s);
        break;
    case ALG_ECSCHNORR_VALUE:
        LoadEccParameter(&testSig.signature.ecschnorrsignatureR, &c_TestEcschnorr_r);
        LoadEccParameter(&testSig.signature.ecschnorrsignatureS, &c_TestEcschnorr_s);
        break;
    case ALG_SM2_VALUE:
        // don't have a test for SM2
        return TPM_RC_SUCCESS;
    default:
        SELF_TEST_FAILURE;
        break;
    }

    TEST_DEFAULT_TEST_HASH(toTest);

    // Have to copy the key. This is because the size used in the test vectors
    // is the size of the ECC parameter for the test key while the size of a point
    // is TPM dependent
    MemoryCopy2B(&testObject.sensitive.sensitive.ecc.b, &c_ecTestKey_ds.b,
        sizeof(testObject.sensitive.sensitive.ecc.t.buffer));
    LoadEccPoint(&testObject.publicArea.unique.ecc, &c_ecTestKey_QsX,
        &c_ecTestKey_QsY);
    testObject.publicArea.parameters.eccDetail.curveID = c_testCurve;

    if (TPM_RC_SUCCESS != CryptEccValidateSignature(&testSig, &testObject,
        (TPM2B_DIGEST *)&c_ecTestValue.b))
    {
        SELF_TEST_FAILURE;
    }

}
CHECK_CANCELED;

// Now sign and verify some data
if(TPM_RC_SUCCESS != CryptEccSign(&testSig, &testObject,
        (TPM2B_DIGEST *)&c_ecTestValue,
        &eccScheme, NULL))
    SELF_TEST_FAILURE;

CHECK_CANCELED;

if(TPM_RC_SUCCESS != CryptEccValidateSignature(&testSig, &testObject,
        (TPM2B_DIGEST *)&c_ecTestValue))
    SELF_TEST_FAILURE;

CHECK_CANCELED;

return TPM_RC_SUCCESS;

10.2.1.6.5  TestKDFa()

static TPM_RC
TestKDFa(
    ALGORITHM_VECTOR   *toTest
)
{
    static TPM2B_KDF_TEST_KEY   keyOut;
    UINT32                      counter = 0;
    // CLEAR_BOTH(ALG_KDF1_SP800_108_VALUE);
    keyOut.t.size = CryptKDFa(KDF_TEST_ALG, &c_kdfTestKeyIn.b, &c_kdfTestLabel.b,
            &c_kdfTestContextU.b, &c_kdfTestContextV.b,
            TEST_KDF_KEY_SIZE * 8, keyOut.t.buffer,
            &counter, FALSE);
    if (   keyOut.t.size != TEST_KDF_KEY_SIZE
        || !MemoryEqual(keyOut.t.buffer, c_kdfTestKeyOut.t.buffer,
            TEST_KDF_KEY_SIZE))
        SELF_TEST_FAILURE;

    return TPM_RC_SUCCESS;

10.2.1.6.6  TestEcc()

static TPM_RC
TestEcc(
    TPM_ALG_ID              alg,
    ALGORITHM_VECTOR        *toTest
)
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    NOT_REFERENCED(toTest);
    switch(alg)
    {
        case ALG_ECC_VALUE:
        case ALG_ECDH_VALUE:
            // If this is in a loop then see if another test is going to deal with
            // this.
            // If toTest is not a self-test list
            if((toTest == &g_toTest)
                // or this is the only ECC test in the list
                || !TEST_BIT(ALG_ECDSA_VALUE, *toTest)
                || TEST_BIT(ALG_ECSPNORM, *toTest)
                || TEST_BIT(ALG_ECSCHNORR, *toTest)
                || TEST_BIT(ALG_ECSNORM, *toTest))
            {
...
10.2.1.6.7 TestAlgorithm()

Dispatches to the correct test function for the algorithm or gets a list of testable algorithms.

If toTest is not NULL, then the test decisions are based on the algorithm selections in toTest. Otherwise, g_toTest is used. When bits are clear in g_toTest they will also be cleared toTest.

If there doesn’t happen to be a test for the algorithm, its associated bit is quietly cleared.

If alg is zero (TPM_ALG_ERROR), then the toTest vector is cleared of any bits for which there is no test (i.e. no tests are actually run but the vector is cleared).

NOTE: toTest will only ever have bits set for implemented algorithms but alg can be anything.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CANCELED</td>
<td>test was canceled</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT
TPM_RC
TestAlgorithm(
    TPM_ALG_ID               alg,
    ALGORITHM_VECTOR        *toTest
)
{
    TPM_ALG_ID              first = (alg == ALG_ERROR_VALUE) ? ALG_FIRST_VALUE : alg;
    TPM_ALG_ID              last = (alg == ALG_ERROR_VALUE) ? ALG_LAST_VALUE : alg;
    BOOL                    doTest = (alg != ALG_ERROR_VALUE);  
    TPM_RC                  result = TPM_RC_SUCCESS;
    if(toTest == NULL)
        toTest = &g_toTest;
    // This is kind of strange. This function will either run a test of the selected
    // algorithm or just clear a bit if there is no test for the algorithm. So,
    // either this loop will be executed once for the selected algorithm or once for
    // each of the possible algorithms. If it is executed more than once ('alg' ==
    // ALG_ERROR), then no test will be run but bits will be cleared for
    // unimplemented algorithms. This was done this way so that there is only one
    // case statement with all of the algorithms. It was easier to have one case
    // statement than to have multiple ones to manage whenever an algorithm ID is
    // added.
    for(alg = first; (alg <= last); alg++)
        { // if 'alg' was TPM_ALG_ERROR, then we will be cycling through
            // values, some of which may not be implemented. If the bit in toTest
```
if(!TEST_BIT(alg, g_implementedAlgorithms))
{
    CLEAR_BIT(alg, *toTest);
    continue;
}

if (!TEST_BIT(alg, g_implementedAlgorithms))
{
    PROCESS_WHATSOEVER_LEFT;
    // NOTE: since this switch will only be called if the algorithm is
    // implemented, it is not necessary to modify this list except to comment
    // out the algorithms for which there is no test
    switch(alg)
    {
        // Symmetric block ciphers
        #if ALG_AES
            case ALG_AES_VALUE:
                // Symmetric modes
                #if !ALG_CFB
                    #error CFB is required in all TPM implementations
                #endif // !ALG_CFB
                case ALG_CFB_VALUE:
                    if(doTest)
                        result = TestSymmetric(alg, toTest);
                    break;
                #if ALG_CTR
                    case ALG_CTR_VALUE:
                    #endif // ALG_CTR
                #if ALG_ECB
                    case ALG_ECB_VALUE:
                    #endif // ALG_ECB
                #if ALG_OFB
                    case ALG_OFB_VALUE:
                    #endif // ALG_OFB
                #if ALG_CBC
                    case ALG_CBC_VALUE:
                    #endif // ALG_CBC
                #if ALG_SM4
                    case ALG_SM4_VALUE:
                    // if SM4 is implemented, its test is like other block ciphers but there
                    // aren't any test vectors for it yet
                    // Symmetric block ciphers
                    #endif // ALG_SM4
                #if ALG_CAMELLIA
                    // no test vectors for camellia
                    case ALG_CAMELLIA_VALUE:
                    #endif // ALG_CAMELLIA
                // Symmetric modes
                #if !ALG_HMAC
                    #error HMAC is required in all TPM implementations
                #endif // !ALG_HMAC
                case ALG_HMAC_VALUE:
                    // Clear the bit that indicates that HMAC is required because
                    // HMAC is used as the basic test for all hash algorithms.
                    CLEAR_BOTH(alg);
                    // Testing HMAC means test the default hash
                    if(doTest)
                        TestHash(DEFAULT_TEST_HASH, toTest);
                    break;
                #if !ALG_HMAC
                    #error HMAC is required in all TPM implementations
                #endif // !ALG_HMAC
                case ALG_HMAC_VALUE:
                    // Clear the bit that indicates that HMAC is required because
                    // HMAC is used as the basic test for all hash algorithms.
                    CLEAR_BOTH(alg);
                    // Testing HMAC means test the default hash
                    if(doTest)
                        TestHash(DEFAULT_TEST_HASH, toTest);
                    else
// If not testing, then indicate that the hash needs to be tested because this uses HMAC
    SET_BOTH(DEFAULT_TEST_HASH);
    break;
}
#endif
#endif
#endif
#endif

if (doTest)
result = TestHash(alg, toTest);
break;
#endif
#endif
#endif
#endif
#endif
#endif
#endif

if (doTest)
result = TestRsa(alg, toTest);
break;
#endif
#endif
#endif
#endif
#endif
#endif
#endif
#endif

if (doTest)
result = TestEcc(alg, toTest);
break;
if (doTest)
    result = TestEcc(alg, toTest);
break;
#endif // ALG_ECC

default:
    CLEAR_BIT(alg, *toTest);
    break;
}

if (result != TPM_RC_SUCCESS)
    break;

return result;
#endif // SELF_TESTS
10.2.2 BnConvert.c

10.2.2.1 Introduction

This file contains the basic conversion functions that will convert TPM2B to/from the internal format. The internal format is a bigNum.

10.2.2.2 Includes

1 #include "Tpm.h"

10.2.2.3 Functions

10.2.2.3.1 BnFromBytes()

This function will convert a big-endian byte array to the internal number format. If bn is NULL, then the output is NULL. If bytes is null or the required size is 0, then the output is set to zero.

2 LIB_EXPORT bigNum
3 BnFromBytes(
4     bigNum           bn,
5     const BYTE      *bytes,
6     NUMBYTES         nBytes
7 )
8 {
9     const BYTE      *pFrom; // 'p' points to the least significant bytes of source
10     BYTE            *pTo;  // points to least significant bytes of destination
11     crypt_uword_t    size;
12     //
13     size = (bytes != NULL) ? BYTES_TO_CRYPT_WORDS(nBytes) : 0;
14     // If nothing in, nothing out
15     if(bn == NULL)
16         return NULL;
17     // make sure things fit
18     pAssert(BnGetAllocated(bn) >= size);
19     if(size > 0)
20     {
21         // Clear the topmost word in case it is not filled with data
22         bn->d[size - 1] = 0;
23         // Moving the input bytes from the end of the list (LSB) end
24         pFrom = bytes + nBytes - 1;
25         // To the LS0 of the LSW of the bigNum.
26         pTo = (BYTE *)bn->d;
27         for(; nBytes != 0; nBytes--)
28             *pTo++ = *pFrom--;
29         // For a little-endian machine, the conversion is a straight byte
30         // reversal. For a big-endian machine, we have to put the words in
31         // big-endian byte order
32         #if BIG_ENDIAN_TPM
33             {
34                 crypt_word_t   t;
35                 for(t = (crypt_word_t)size - 1; t >= 0; t--)
36                     bn->d[t] = SWAP_CRYPT_WORD(bn->d[t]);
37             }
38         #endif
39     }
40 }
10.2.2.3.2 BnFrom2B()

Convert an TPM2B to a BIG_NUM. If the input value does not exist, or the output does not exist, or the input will not fit into the output the function returns NULL.

```c
LIB_EXPORT bigNum
BnFrom2B(
    bigNum       bn,           // OUT:
    const TPM2B *a2B       // IN: number to convert
)
{
    if(a2B != NULL)
        return BnFromBytes(bn, a2B->buffer, a2B->size);
    // Make sure that the number has an initialized value rather than whatever
    // was there before
    BnSetTop(bn, 0);    // Function accepts NULL
    return NULL;
}
```

10.2.2.3.3 BnFromHex()

Convert a hex string into a bigNum. This is primarily used in debugging.

```c
LIB_EXPORT bigNum
BnFromHex(
    bigNum       bn,       // OUT:
    const char *hex       // IN:
)
{
    #define FromHex(a)  ((a) - (((a) > 'a') ? ('a' + 10) : ('a' - 10))
    unsigned i;
    unsigned wordCount;
    const char *p;
    BYTE                *d = (BYTE *)&(bn->d[0]);
    // pAssert(bn && hex);
    i = strlen(hex);
    wordCount = BYTES_TO_CRYPT_WORDS((i + 1) / 2);
    if((i == 0) || (wordCount >= BnGetAllocated(bn)))
        BnSetWord(bn, 0);
    else
    {
        bn->d[wordCount - 1] = 0;
        p = hex + i - 1;
        for(; i > 1; i -= 2)
            {
                BYTE a;
                a = FromHex(*p);
                p--;
                *d++ = a + (FromHex(*p) << 4);
                p--;
            }
        if(i == 1)
            *d = FromHex(*p);
    }
    #if !BIG_ENDIAN_TPM
    for(i = 0; i < wordCount; i++)
        bn->d[i] = SWAP_CRYPT_WORD(bn->d[i]);
    #endif
}
```
10.2.2.3.4 BnToBytes()

This function converts a BIG_NUM to a byte array. It converts the bigNum to a big-endian byte string and sets size to the normalized value. If size is an input 0, then the receiving buffer is guaranteed to be large enough for the result and the size will be set to the size required for bigNum (leading zeros suppressed).

The conversion for a little-endian machine simply requires that all significant bytes of the bigNum be reversed. For a big-endian machine, rather than unpack each word individually, the bigNum is converted to little-endian words, copied, and then converted back to big-endian.

```c
LIB_EXPORT BOOL BnToBytes(
    bigConst             bn,
    BYTE                *buffer,
    NUMBYTES            *size
) {
    crypt_uword_t        requiredSize;
    BYTE                *pFrom;
    BYTE                *pTo;
    crypt_uword_t        count;
    // validate inputs
    pAssert(bn && buffer && size);

    requiredSize = (BnSizeInBits(bn) + 7) / 8;
    if(requiredSize == 0)
    {
        // If the input value is 0, return a byte of zero
        *size = 1;
        *buffer = 0;
    }
    else
    {
        #if BIG_ENDIAN_TPM
        // Copy the constant input value into a modifiable value
        BN_VAR(bnL, LARGEST_NUMBER_BITS * 2);
        BnCopy(bnL, bn);
        // byte swap the words in the local value to make them little-endian
        for(count = 0; count < bnL->size; count++)
        {
            bnL->d[count] = SWAP_CRYPT_WORD(bnL->d[count]);
        }
        bn = (bigConst)bnL;
        #endif
        if(*size == 0)
        {
            *size = (NUMBYTES)requiredSize;
            pAssert(requiredSize <= *size);
            // Byte swap the number (not words but the whole value)
            count = *size;
            // Start from the least significant word and offset to the most significant
            // byte which is in some high word
            pFrom = (BYTE *)(&bn->d[0]) + requiredSize - 1;
            pTo = buffer;

            // If the number of output bytes is larger than the number bytes required
            // for the input number, pad with zeros
            for(count = *size; count > requiredSize; count--)
            {
                *pTo++ = 0;
            }
        }
    }
    return bn;
}
```
// Move the most significant byte at the end of the BigNum to the next most
// significant byte position of the 2B and repeat for all significant bytes.
for(; requiredSize > 0; requiredSize--)
    *pTo++ = *pFrom--;
return TRUE;
}

10.2.2.3.5 BnTo2B()

Function to convert a BIG_NUM to TPM2B. The TPM2B size is set to the requested size which may require padding. If size is non-zero and less than required by the value in bn then an error is returned. If size is zero, then the TPM2B is assumed to be large enough for the data and a2b->size will be adjusted accordingly.

LIB_EXPORT BOOL
BnTo2B{
    bigConst bn, // IN:
    TPM2B *a2B, // OUT:
    NUMBYTES size // IN: the desired size
    }
    // Set the output size
    if(bn && a2B)
    {
        a2B->size = size;
        return BnToBytes(bn, a2B->buffer, &a2B->size);
    }
    return FALSE;
}

10.2.2.3.6 BnPointFrom2B()

Function to create a BIG_POINT structure from a 2B point. A point is going to be two ECC values in the same buffer. The values are going to be the size of the modulus. They are in modular form.

LIB_EXPORT bn_point_t *
BnPointFrom2B{
    bigPoint ecP, // OUT: the preallocated point structure
    TPMs_ECC_POINT *p // IN: the number to convert
    }
    if(p == NULL)
    return NULL;
    if(NULL != ecP)
    {
        BnFrom2B(ecP->x, &p->x.b);
        BnFrom2B(ecP->y, &p->y.b);
        BnSetWord(ecP->z, 1);
    }
    return ecP;
}

10.2.2.3.7 BnPointTo2B()

This function converts a BIG_POINT into a TPMs_ECC_POINT. A TPMs_ECC_POINT contains two TPM2B_ECC_PARAMETER values. The maximum size of the parameters is dependent on the maximum
EC key size used in an implementation. The presumption is that the TPMS_ECC_POINT is large enough to hold 2 TPM2B values, each as large as a MAX_ECC_PARAMETER_BYTES

189   LIB_EXPORT BOOL
190   BnPointTo2B(
191       TPMS_ECC_POINT *p,      // OUT: the converted 2B structure
192       bigPoint ecP,          // IN: the values to be converted
193       bigCurve E            // IN: curve descriptor for the point
194   )
195
196   {  //
197       UINT16 size;
198
199       pAssert(p && ecP && E);
200       pAssert(BnEqualWord(ecP->z, 1));
201       // BnMsb is the bit number of the MSB. This is one less than the number of bits
202       size = (UINT16)BITS_TO_BYTES(BnSizeInBits(CurveGetOrder(AccessCurveData(E))));
203       BnTo2B(ecP->x, &p->x.b, size);
204       BnTo2B(ecP->y, &p->y.b, size);
205       return TRUE;
206   }
207#endif  // ALG_ECC
10.2.3  BnMath.c

10.2.3.1  Introduction

The simulator code uses the canonical form whenever possible in order to make the code in Part 3 more accessible. The canonical data formats are simple and not well suited for complex big number computations. When operating on big numbers, the data format is changed for easier manipulation. The format is native words in little-endian format. As the magnitude of the number decreases, the length of the array containing the number decreases but the starting address doesn’t change.

The functions in this file perform simple operations on these big numbers. Only the more complex operations are passed to the underlying support library. Although the support library would have most of these functions, the interface code to convert the format for the values is greater than the size of the code to implement the functions here. So, rather than incur the overhead of conversion, they are done here.

If an implementer would prefer, the underlying library can be used simply by making code substitutions here.

NOTE: There is an intention to continue to augment these functions so that there would be no need to use an external big number library.

Many of these functions have no error returns and will always return TRUE. This is to allow them to be used in guarded sequences. That is: OK = OK || BnSomething(s); where the BnSomething() function should not be called if OK isn’t true.

10.2.3.2  Includes

```c
#include "Tpm.h"
```

A constant value of zero as a stand in for NULL bigNum values

```c
const bignum_t BnConstZero = {1, 0, {0}};
```

10.2.3.3  Functions

10.2.3.3.1  AddSame()

Add two values that are the same size. This function allows result to be the same as either of the addends. This is a nice function to put into assembly because handling the carry for multi-precision stuff is not as easy in C (unless there is a REALLY smart compiler). It would be nice if there were idioms in a language that a compiler could recognize what is going on and optimize loops like this.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no carry out</td>
</tr>
<tr>
<td>1</td>
<td>carry out</td>
</tr>
</tbody>
</table>

```c
static BOOL AddSame(
    crypt_uword_t *result,
    const crypt_uword_t *op1,
    const crypt_uword_t *op2,
    int count
)
{
    int carry = 0;
    int i;
```
Trusted Platform Module Library

Part 4: Supporting Routines

for (i = 0; i < count; i++)
{

crypt_uword_t a = op1[i];
crypt_uword_t sum = a + op2[i];
result[i] = sum + carry;
// generate a carry if the sum is less than either of the inputs
// do this using bit operations rather than logical operations so that
// the time is about the same.
//             propagate term      | generate term
carry = ((result[i] == 0) & carry) | (sum < a);
}
return carry;

10.2.3.3.2 CarryProp()

Propagate a carry

static int CarryProp(
crypt_uword_t *result,
const crypt_uword_t *op,
int count,
int carry )
{
for (; count; count--)
carry = ((*result++) = *op++ + carry) == 0) & carry;
return carry;
}

10.2.3.3.3 BnAdd()

This function adds two bigNum values. This function always returns TRUE.

LIB_EXPORT BOOL BnAdd(
bigNum result,
bigConst op1,
bigConst op2 )
{
crypt_uword_t stop;
int carry;
const bignum_t *n1 = op1;
const bignum_t *n2 = op2;
if(n2->size > n1->size)
{
    n1 = op2;
n2 = op1;
}
pAssert(result->allocated >= n1->size);
stop = MIN(n1->size, n2->allocated);
carry = AddSame(result->d, n1->d, n2->d, stop);
if(n1->size > stop)
carry = CarryProp(&result->d[stop], &n1->d[stop], n1->size - stop, carry);
CarryResolve(result, n1->size, carry);
return TRUE;

10.2.3.3.4 BnAddWord()

This function adds a word value to a bigNum. This function always returns TRUE.

LIB_EXPORT BOOL  
BnAddWord(  
bigNum           result,  
bigConst         op,  
crypt_uword_t    word  
)
{
    int    carry;
    //
    carry = (result->d[0] = op->d[0] + word) < word;
    carry = CarryProp(&result->d[1], &op->d[1], op->size - 1, carry);
    CarryResolve(result, op->size, carry);
    return TRUE;
}

10.2.3.3.5 SubSame()

This function subtracts two values that have the same size.

static int  
SubSame(  
crypt_uword_t *result,  
const crypt_uword_t *op1,  
const crypt_uword_t *op2,  
int    count  
)
{
    int      borrow = 0;
    int      i;
    for(i = 0; i < count; i++)
    {
        crypt_uword_t a = op1[i];
crypt_uword_t diff = a - op2[i];
        result[i] = diff - borrow;
        // generate | propagate
        borrow = (diff > a) | ((diff == 0) & borrow);
    }
    return borrow;
}
10.2.3.3.6 BorrowProp()

This propagates a borrow. If borrow is true when the end of the array is reached, then it means that op2 was larger than op1 and we don't handle that case so an assert is generated. This design choice was made because our only bigNum computations are on large positive numbers (primes) or on fields.

Propagate a borrow.

```
static int BorrowProp(
    crypt_uword_t *result,
    const crypt_uword_t *op,
    int size,
    int borrow
){
    for(; size > 0; size--)
        borrow = ((*result++ = *op++ - borrow) == MAX_CRYPT_UWORD) && borrow;
    return borrow;
}
```

10.2.3.3.7 BnSub()

This function does subtraction of two bigNum values and returns result = op1 - op2 when op1 is greater than op2. If op2 is greater than op1, then a fault is generated. This function always returns TRUE.

```
LIBEXPORT BOOL BnSub(
    bigNum result,
    bigConst op1,
    bigConst op2
){
    int borrow;
    crypt_uword_t stop = MIN(op1->size, op2->allocated);
    // Make sure that op2 is not obviously larger than op1
    pAssert(op1->size >= op2->size);
    borrow = SubSame(result->d, op1->d, op2->d, stop);
    if(op1->size > stop)
        borrow = BorrowProp(&result->d[stop], &op1->d[stop], op1->size - stop, borrow);
    pAssert(!borrow);
    BnSetTop(result, op1->size);
    return TRUE;
}
```

10.2.3.3.8 BnSubWord()

This function subtracts a word value from a bigNum. This function always returns TRUE.

```
LIBEXPORT BOOL BnSubWord(
    bigNum result,
    bigConst op,
    crypt_uword_t word
){
    int borrow;
    // pAssert(op->size > 1 || word <= op->d[0]);
    borrow = word > op->d[0];
}
```
157    result->d[0] = op->d[0] - word;
158    borrow = BorrowProp(&result->d[1], &op->d[1], op->size - 1, borrow);
159    pAssert(!borrow);
160    BnSetTop(result, op->size);
161    return TRUE;
162 }

10.2.3.3.9 BnUnsignedCmp()

This function performs a comparison of op1 to op2. The compare is approximately constant time if the
size of the values used in the compare is consistent across calls (from the same line in the calling code).

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>op1 is less than op2</td>
</tr>
<tr>
<td>0</td>
<td>op1 is equal to op2</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>op1 is greater than op2</td>
</tr>
</tbody>
</table>

163 LIB_EXPORT int
164 BnUnsignedCmp(
165     bigConst op1,
166     bigConst op2
167 )
168 {
169     int    retVal;
170     int    diff;
171     int    i;
172     //
173     pAssert((op1 != NULL) && (op2 != NULL));
174     retVal = op1->size - op2->size;
175     if(retVal == 0)
176     {
177         for(i = (int)(op1->size - 1); i >= 0; i--)
178             { diff = (op1->d[i] < op2->d[i]) ? -1 : (op1->d[i] != op2->d[i]);
179                  retVal = retVal == 0 ? diff : retVal;
180             }
181     }
182     else
183         retVal = (retVal < 0) ? -1 : 1;
184     return retVal;
185 }

10.2.3.3.10 BnUnsignedCmpWord()

Compare a bigNum to a crypt_uword_t.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>op1 is less than word</td>
</tr>
<tr>
<td>0</td>
<td>op1 is equal to word</td>
</tr>
<tr>
<td>1</td>
<td>op1 is greater than word</td>
</tr>
</tbody>
</table>

187 LIB_EXPORT int
188 BnUnsignedCmpWord(
189     bigConst op1,
190     crypt_uword_t word
191 )
192 {
if (op1->size > 1)
    return 1;
else if (op1->size == 1)
    return (op1->d[0] < word) ? -1 : (op1->d[0] > word);
else // op1 is zero
    // equal if word is zero
    return (word == 0) ? 0 : -1;
}

10.2.3.3.11 BnModWord()

This function does modular division of a big number when the modulus is a word value.

LIB_EXPORT crypt_word_t BnModWord(
    const crypt_uword_t* bigConst,         // numerator
    crypt_word_t modulus                    // modulus
)
{
    BN_MAX(remainder);
    BN_VAR(mod, RADIX_BITS);
    mod->d[0] = modulus;
    mod->size = (modulus != 0);
    BnDiv(NULL, remainder, numerator, mod);
    return remainder->d[0];
}

10.2.3.3.12 Msb()

This function returns the bit number of the most significant bit of a crypt_uword_t. The number for the least significant bit of any bigNum value is 0. The maximum return value is RADIX_BITS - 1.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>the word was zero</td>
</tr>
<tr>
<td>n</td>
<td>the bit number of the most significant bit in the word</td>
</tr>
</tbody>
</table>

LIB_EXPORT int Msb(
    const crypt_uword_t* word
)
{
    int retVal = -1;
    #if RADIX_BITS == 64
    if (word & 0xffffffff00000000) { retVal += 32; word >>= 32; }
    #endif
    if (word & 0xffff0000) { retVal += 16; word >>= 16; }
    if (word & 0x0000ff00) { retVal += 8; word >>= 8; }
    if (word & 0x000000f0) { retVal += 4; word >>= 4; }
    if (word & 0x0000000c) { retVal += 2; word >>= 2; }
    if (word & 0x00000002) { retVal += 1; word >>= 1; }
    return retVal + (int)word;
}

10.2.3.3.13 BnMsb()

This function returns the number of the MSb() of a bigNum value.
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>the word was zero or bn was NULL</td>
</tr>
<tr>
<td>n</td>
<td>the bit number of the most significant bit in the word</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT int BnMsb(
    bigConst bn
) {
    // If the value is NULL, or the size is zero then treat as zero and return -1
    if(bn != NULL && bn->size > 0)
    {
        int retVal = Msb(bn->d[bn->size - 1]);
        retVal += (bn->size - 1) * RADIX_BITS;
        return retVal;
    }
    else
    { return -1; }
}
```

### 10.2.3.3.14 BnSizeInBits()

This function returns the number of bits required to hold a number. It is one greater than the Msb.

```c
LIB_EXPORT unsigned BnSizeInBits(
    bigConst n
) {
    int bits = BnMsb(n) + 1;
    // return bits < 0? 0 : (unsigned)bits;
}
```

### 10.2.3.3.15 BnSetWord()

Change the value of a bigNum_t to a word value.

```c
LIB_EXPORT bigNum BnSetWord(
    bigNum n,
    crypt_uword_t w
) {
    if(n != NULL)
    {
        pAssert(n->allocated > 1);
        n->d[0] = w;
        BnSetTop(n, (w != 0) ? 1 : 0);
        return n;
    }
}
```

### 10.2.3.3.16 BnSetBit()

This function will SET a bit in a bigNum. Bit 0 is the least-significant bit in the 0th digit_t. The function always return TRUE.

```c
LIB_EXPORT BOOL BnSetBit(
    bigNum bigNum,
    digit_t digit,
    crypt_uword_t w
) {
    if(w)
    {
        BnSetTop(bigNum, digit, 1);
        BnSetTop(bigNum, digit, (w != 0) ? 1 : 0);
        return true;
    }
    return false;
}
```
BnSetBit(
bigNum           bn,       // IN/OUT: big number to modify
unsigned int    bitNum    // IN: Bit number to SET
)
{
    crypt_uword_t offset = bitNum / RADIX_BITS;
pAssert(bn->allocated * RADIX_BITS >= bitNum);
    // Grow the number if necessary to set the bit.
    while(bn->size <= offset)
        bn->d[bn->size++] = 0;
    bn->d[offset] |= (1 << RADIX_MOD(bitNum));
    return TRUE;
}

10.2.3.3.17 BnTestBit()

This function is used to check to see if a bit is SET in a bignum_t. The 0th bit is the LSB() of d[0].

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>the bit is set</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>the bit is not set or the number is out of range</td>
</tr>
</tbody>
</table>

BnTestBit(
bigNum           bn,       // IN: number to check
unsigned int    bitNum    // IN: bit to test
)
{
    crypt_uword_t offset = RADIX_DIV(bitNum);
    //
    if(bn->size > offset)
        return ((bn->d[offset] & (((crypt_uword_t)1) << RADIX_MOD(bitNum))) != 0);
    else
        return FALSE;
}

10.2.3.3.18 BnMaskBits()

This function is used to mask off high order bits of a big number. The returned value will have no more than maskBit bits set.

NOTE: There is a requirement that unused words of a bignum_t are set to zero.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>result masked</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>the input was not as large as the mask</td>
</tr>
</tbody>
</table>

BnMaskBits(
bigNum           bn,       // IN/OUT: number to mask
crypt_uword_t    maskBit    // IN: the bit number for the mask.
)
{
    crypt_uword_t finalSize;
    BOOL             retVal;
    finalSize = BITS_TO_CRYPT_WORDS(maskBit);
    retVal = (finalSize <= bn->allocated);
if (retVal && (finalSize > 0))
{
    crypt_uword_t    mask;
    mask = ~((crypt_uword_t)0) >> RADIX_MOD(maskBit);
    bn->d[finalSize - 1] &= mask;
}
BnSetTop(bn, finalSize);
return retVal;

10.2.3.3.19 BnShiftRight()

This function will shift a bigNum to the right by the shiftAmount. This function always returns TRUE.

LIB_EXPORT BOOL
BnShiftRight(
    bigNum       result,
    bigConst     toShift,
    uint32_t     shiftAmount
)
{
    uint32_t    offset = (shiftAmount >> RADIX_LOG2);
    uint32_t    i;
    uint32_t    shiftIn;
    crypt_uword_t    finalSize;

    // shiftAmount = shiftAmount & RADIX_MASK;
    shiftIn = RADIX_BITS - shiftAmount;
    // The end size is toShift->size - offset less one additional
    // word if the shiftAmount would make the upper word == 0
    if(toShift->size > offset)
    {
        finalSize = toShift->size - offset;
        finalSize -= (toShift->d[toShift->size - 1] >> shiftAmount) == 0 ? 1 : 0;
    }
    else
    {
        finalSize = 0;
    }
    pAssert(finalSize <= result->allocated);
    if(finalSize != 0)
    {
        for(i = 0; i < finalSize; i++)
        {
            result->d[i] = (toShift->d[i + offset] >> shiftAmount)
                             | (toShift->d[i + offset + 1] << shiftIn);
        }
        if(offset == 0)
            result->d[i] = toShift->d[i] >> shiftAmount;
    }
    BnSetTop(result, finalSize);
    return TRUE;

10.2.3.3.20 BnGetRandomBits()

This function gets random bits for use in various places. To make sure that the number is generated in a portable format, it is created as a TPM2B and then converted to the internal format.

One consequence of the generation scheme is that, if the number of bits requested is not a multiple of 8, then the high-order bits are set to zero. This would come into play when generating a 521-bit ECC key. A 66-byte (528-bit) value is generated an the high order 7 bits are masked off (CLEAR).
### Return Value

<table>
<thead>
<tr>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>success</td>
</tr>
<tr>
<td>failure</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT BOOL

BnGetRandomBits(bigNum, n, size_t bits, RAND_STATE *rand) {
    // Since this could be used for ECC key generation using the extra bits method,
    // make sure that the value is large enough
    TPM2B_TYPE(LARGEST, LARGEST_NUMBER + 8);
    TPM2B_LARGEST large;
    large.b.size = (UINT16)BITS_TO_BYTES(bits);
    if (DRBG_Generate(rand, large.t.buffer, large.t.size) == large.t.size) {
        if (BnFrom2B(n, &large.b) != NULL) {
            if (BnMaskBits(n, bits)) return TRUE;
        }
    }
    return FALSE;
}
```

10.2.3.3.21 BnGenerateRandomInRange()

This function is used to generate a random number \( r \) in the range \( 1 \leq r < \text{limit} \). The function gets a random number of bits that is the size of \( \text{limit} \). There is some probability that the returned number is going to be greater than or equal to the \( \text{limit} \). If it is, try again. There is no more than 50% chance that the next number is also greater, so try again. We keep trying until we get a value that meets the criteria. Since \( \text{limit} \) is very often a number with a LOT of high order ones, this rarely would need a second try.

```c
LIB_EXPORT BOOL

BnGenerateRandomInRange(bigNum, dest, bigConst limit, RAND_STATE *rand) {
    size_t bits = BnSizeInBits(limit);
    if (bits < 2) {
        BnSetWord(dest, 0);
        return FALSE;
    }
    else {
        do
        BnGetRandomBits(dest, bits, rand);
    }
```
398     } while(BnEqualZero(dest) || BnUnsignedCmp(dest, limit) >= 0);
399   }
400   return TRUE;
401 }
10.2.4 BnMemory.c

10.2.4.1 Introduction

This file contains the memory setup functions used by the bigNum functions in CryptoEngine().

10.2.4.2 Includes

1 #include "Tpm.h"

10.2.4.3 Functions

10.2.4.3.1 BnSetTop()

This function is used when the size of a bignum_t is changed. It makes sure that the unused words are set to zero and that any significant words of zeros are eliminated from the used size indicator.

2 LIB_EXPORT bigNum
3 BnSetTop(
4     bigNum     bn,       // IN/OUT: number to clean
5     crypt_uword_t    top      // IN: the new top
6     )
7     {
8         if(bn != NULL)
9             {
10                pAssert(top <= bn->allocated);
11                // If forcing the size to be decreased, make sure that the words being
12                // discarded are being set to 0
13                while(bn->size > top)
14                    bn->d[--bn->size] = 0;
15                bn->size = top;
16                // Now make sure that the words that are left are 'normalized' (no high-order
17                // words of zero.
18                while((bn->size > 0) && (bn->d[bn->size - 1] == 0))
19                    bn->size -= 1;
20            }
21         return bn;
22     }

10.2.4.3.2 BnClearTop()

This function will make sure that all unused words are zero.

23 LIB_EXPORT bigNum
24 BnClearTop(
25     bigNum     bn
26     )
27     {
28         crypt_uword_t    i;
29         //
30         if(bn != NULL)
31             {
32                for(i = bn->size; i < bn->allocated; i++)
33                    bn->d[i] = 0;
34                while((bn->size > 0) && (bn->d[bn->size] == 0))
35                    bn->size -= 1;
36             }
37         return bn;
38     }
10.2.4.3.3 BnInitializeWord()

This function is used to initialize an allocated bigNum with a word value. The bigNum does not have to be allocated with a single word.

```c
LIB_EXPORT bigNum
BnInitializeWord(
    bigNum      bn,       // IN:
    crypt_uword_t allocated, // IN:
    crypt_uword_t word      // IN:
)
{
    bn->allocated = allocated;
    bn->size = (word != 0);
    bn->d[0] = word;
    while(allocated > 1)
        bn->d[--allocated] = 0;
    return bn;
}
```

10.2.4.3.4 BnInit()

This function initializes a stack allocated bignum_t. It initializes allocated and size and zeros the words of d.

```c
LIB_EXPORT bigNum
BnInit(
    bigNum               bn,
    crypt_uword_t        allocated
)
{
    if(bn != NULL)
    {
        bn->allocated = allocated;
        bn->size = 0;
        while(allocated != 0)
            bn->d[--allocated] = 0;
        return bn;
    }
}
```

10.2.4.3.5 BnCopy()

Function to copy a bignum_t. If the output is NULL, then nothing happens. If the input is NULL, the output is set to zero.

```c
LIB_EXPORT BOOL
BnCopy(  
    bigNum           out,
    bigConst         in
)
{
    if(in == out)
        BnSetTop(out, BnGetSize(out));
    else if(out != NULL)
    {
        if(in != NULL)
        {
            unsigned int i;
            pAssert(BnGetAllocated(out) >= BnGetSize(in));
            for(i = 0; i < BnGetSize(in); i++)
                out->d[i] = in->d[i];
        }
    }
}
84       BnSetTop(out, BnGetSize(in));
85     }
86     else
87       BnSetTop(out, 0);
88   }
89   return TRUE;
90 }
91 #if ALG_ECC

10.2.4.3.6 BnPointCopy()

Function to copy a bn point.

92 LIBEXPORT BOOL
93 BnPointCopy(
94    bigPoint pOut,
95    pointConst pIn
96 )
97 {
98   return BnCopy(pOut->x, pIn->x)
99      && BnCopy(pOut->y, pIn->y)
100     && BnCopy(pOut->z, pIn->z);
101 }

10.2.4.3.7 BnInitializePoint()

This function is used to initialize a point structure with the addresses of the coordinates.

102 LIBEXPORT bn_point_t *
103 BnInitializePoint(
104    bigPoint p,
105    bigNum x,
106    bigNum y,
107    bigNum z
108 )
109 {
110   p->x = x;
111   p->y = y;
112   p->z = z;
113   BnSetWord(z, 1);
114   return p;
115 }
116 #endif // ALG_ECC
10.2.5 CryptCmac.c

10.2.5.1 Introduction

This file contains the implementation of the message authentication codes based on a symmetric block cipher. These functions only use the single block encryption functions of the selected symmetric cryptographic library.

10.2.5.2 Includes, Defines, and Typedefs

```c
#define _CRYPT_HASH_C_
#include "Tpm.h"
#include "CryptSym.h"
#if ALG_CMAC
```

10.2.5.3 Functions

10.2.5.3.1 CryptCmacStart()

This is the function to start the CMAC sequence operation. It initializes the dispatch functions for the data and end operations for CMAC and initializes the parameters that are used for the processing of data, including the key, key size and block cipher algorithm.

```c
UINT16 CryptCmacStart(
    SMAC_STATE *state,
    TPMU_PUBLIC_PARMS *keyParms,
    TPM_ALG_ID macAlg,
    TPM2B *key
)
{
    tpmCmacState_t *cState = &state->state.cmac;
    TPMT_SYM_DEF_OBJECT *def = &keyParms->symDetail.sym;
    // if(macAlg != TPM_ALG_CMAC)
    return 0;
    // set up the encryption algorithm and parameters
    cState->symAlg = def->algorithm;
    cState->keySizeBits = def->keyBits.sym;
    cState->iv.t.size = CryptGetSymmetricBlockSize(def->algorithm,
        def->keyBits.sym);
    MemoryCopy2B(&cState->symKey.b, key, sizeof(cState->symKey.t.buffer));
    // Set up the dispatch methods for the CMAC
    state->smacMethods.data = CryptCmacData;
    state->smacMethods.end = CryptCmacEnd;
    return cState->iv.t.size;
}
```

10.2.5.3.2 CryptCmacData()

This function is used to add data to the CMAC sequence computation. The function will XOR new data into the IV. If the buffer is full, and there is additional input data, the data is encrypted into the IV buffer, the new data is then XOR into the IV. When the data runs out, the function returns without encrypting even if the buffer is full. The last data block of a sequence will not be encrypted until the call to CryptCmacEnd(). This is to allow the proper subkey to be computed and applied before the last block is encrypted.
void CryptCmacData(
    SMAC_STATES *state,
    UINT32 size,
    const BYTE *buffer
)
{
    tpmCmacState_t *cmacState = &state->cmac;
    TPM_ALG_ID algorithm = cmacState->symAlg;
    BYTE *key = cmacState->symKey.t.buffer;
    UINT16 keySizeInBits = cmacState->keySizeBits;
    tpmCryptKeySchedule_t keySchedule;
    TpmCryptSetSymKeyCall_t encrypt;
    //
    SELECT(ENCRYPT);
    while(size > 0)
    {
        if(cmacState->bcount == cmacState->iv.t.size)
        {
            ENCRYPT(&keySchedule, cmacState->iv.t.buffer, cmacState->iv.t.buffer);
            cmacState->bcount = 0;
        }
        for(;(size > 0) && (cmacState->bcount < cmacState->iv.t.size);
            size--, cmacState->bcount++)
        {
            cmacState->iv.t.buffer[cmacState->bcount] ^= *buffer++;
        }
    }
}

10.2.5.3.3 CryptCmacEnd()

This is the completion function for the CMAC. It does padding, if needed, and selects the subkey to be applied before the last block is encrypted.

UINT16 CryptCmacEnd(
    SMAC_STATES *state,
    UINT32 outSize,
    BYTE *outBuffer
)
{
    tpmCmacState_t *cState = &state->cmac;
    // Need to set algorithm, key, and keySizeInBits in the local context so that
    // the SELECT and ENCRYPT macros will work here
    TPM_ALG_ID algorithm = cState->symAlg;
    BYTE *key = cState->symKey.t.buffer;
    UINT16 keySizeInBits = cState->keySizeBits;
    tpmCryptKeySchedule_t keySchedule;
    TpmCryptSetSymKeyCall_t encrypt;
    TPM2B_IV subkey = {{0, {0}}};
    BOOL xorVal;
    UINT16 i;
    subkey.t.size = cState->iv.t.size;
    // Encrypt a block of zero
    SELECT(ENCRYPT);
    ENCRYPT(&keySchedule, subkey.t.buffer, subkey.t.buffer);
    // shift left by 1 and XOR with 0x0...87 if the MSb was 0
    xorVal = ((subkey.t.buffer[0] & 0x80) == 0) ? 0 : 0x87;
    ShiftLeft(&subkey.b);
    subkey.t.buffer[subkey.t.size - 1] ^= xorVal;
    // this is a sanity check to make sure that the algorithm is working properly.
88  // remove this check when debug is done
89  pAssert(cState->bcount <= cState->iv.t.size);
90  // If the buffer is full then no need to compute subkey 2.
91  if(cState->bcount < cState->iv.t.size)
92  {
93      //Pad the data
94      cState->iv.t.buffer[cState->bcount++] ^= 0x80;
95      // The rest of the data is a pad of zero which would simply be XORed
96      // with the iv value so nothing to do...
97      // Now compute K2
98      xorVal = ((subkey.t.buffer[0] & 0x80) == 0) ? 0 : 0x87;
99      ShiftLeft(&subkey.b);
100     subkey.t.buffer[subkey.t.size - 1] ^= xorVal;
101  }
102  // XOR the subkey into the IV
103  for(i = 0; i < subkey.t.size; i++)
104      cState->iv.t.buffer[i] ^= subkey.t.buffer[i];
105  ENCRYPT(&keySchedule, cState->iv.t.buffer, cState->iv.t.buffer);
106  i = (UINT16)MIN(cState->iv.t.size, outSize);
107  MemoryCopy(outBuffer, cState->iv.t.buffer, i);
108  return i;
109 }
10.2.6 CryptUtil.c

10.2.6.1 Introduction

This module contains the interfaces to the CryptoEngine() and provides miscellaneous cryptographic functions in support of the TPM.

10.2.6.2 Includes

```c
#include "Tpm.h"
```

10.2.6.3 Hash/HMAC Functions

10.2.6.3.1 CryptHmacSign()

Sign a digest using an HMAC key. This an HMAC of a digest, not an HMAC of a message.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HASH</td>
<td>not a valid hash</td>
</tr>
</tbody>
</table>

```c
static TPM_RC CryptHmacSign(TPMT_SIGNATURE *signature, // OUT: signature
               OBJECT *signKey, // IN: HMAC key sign the hash
               TPM2B_DIGEST *hashData // IN: hash to be signed
               )
{
    HMAC_STATE hmacState;
    UINT32 digestSize;
    digestSize = CryptHmacStart2B(&hmacState, signature->signature.any.hashAlg,
       &signKey->sensitive.sensitive.bits.b);
    CryptDigestUpdate2B(&hmacState.hashState, &hashData->b);
    CryptHmacEnd(&hmacState, digestSize,
       (BYTE *)&signature->signature.hmac.digest);
    return TPM_RC_SUCCESS;
}
```

10.2.6.3.2 CryptHMACVerifySignature()

This function will verify a signature signed by a HMAC key. Note that a caller needs to prepare `signature` with the signature algorithm (TPM_ALG_HMAC) and the hash algorithm to use. This function then builds a signature of that type.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SCHEME</td>
<td>not the proper scheme for this key type</td>
</tr>
<tr>
<td>TPM_RC_SIGNATURE</td>
<td>if invalid input or signature is not genuine</td>
</tr>
</tbody>
</table>

```c
static TPM_RC CryptHMACVerifySignature(OBJECT *signKey, // IN: HMAC key signed the hash
               TPM2B_DIGEST *hashData, // IN: digest being verified
               TPMT_SIGNATURE *signature // IN: signature to be verified
               )
{
```
TPMT_SIGNATURE test;
TPMT_KEYEDHASH_SCHEME *keyScheme =
    &signKey->publicArea.parameters.keyedHashDetail.scheme;

//
if((signature->sigAlg != ALG_HMAC_VALUE)
    || (signature->signature.hmac.hashAlg == ALG_NULL_VALUE))
    return TPM_RC_SCHEME;
// This check is not really needed for verification purposes. However, it does
// prevent someone from trying to validate a signature using a weaker hash
// algorithm than otherwise allowed by the key. That is, a key with a scheme
// other than TMP_ALG_NULL can only be used to validate signatures that have
// a matching scheme.
if((keyScheme->scheme != ALG_NULL_VALUE)
    && ((keyScheme->scheme != signature->sigAlg)
        || (keyScheme->details.hmac.hashAlg
            != signature->signature.any.hashAlg)))
    return TPM_RC_SIGNATURE;
test.sigAlg = signature->sigAlg;
test.signature.hmac.hashAlg = signature->signature.hmac.hashAlg;
CryptHmacSign(&test, signKey, hashData);

// Compare digest
if(!MemoryEqual(&test.signature.hmac.digest,
    &signature->signature.hmac.digest, CryptHashGetDigestSize(signature->signature.any.hashAlg)))
    return TPM_RC_SIGNATURE;

return TPM_RC_SUCCESS;

10.2.6.3.3 CryptGenerateKeyedHash()

This function creates a keyedHash object.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>cannot get values from random number generator</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>sensitive data size is larger than allowed for the scheme</td>
</tr>
</tbody>
</table>

static TPM_RC
CryptGenerateKeyedHash(
    TPMT_PUBLIC *publicArea, // IN/OUT: the public area template
    TPMTSENSITIVE *sensitive, // for the new key.
    TPMSSENSITIVE_CREATE *sensitiveCreate, // OUT: sensitive area
    RANDSTATE *rand // IN: sensitive creation data
) {

    TPMT_KEYEDHASH_SCHEME *scheme;
    TPM_ALG_ID hashAlg;
    UINT16 blockSize;
    UINT16 digestSize;

    scheme = &publicArea->parameters.keyedHashDetail.scheme;
    if(publicArea->type != ALG_KEYEDHASH_VALUE)
        return TPM_RC_FAILURE;

    // Pick the limiting hash algorithm
    if(scheme->scheme == ALG_NULL_VALUE)
        hashAlg = publicArea->nameAlg;
    else if(scheme->scheme == ALG_XOR_VALUE)
hashAlg = scheme->details.xor.hashAlg;
else
  hashAlg = scheme->details.hmac.hashAlg;
hashBlockSize = CryptHashGetBlockSize(hashAlg);
digestSize = CryptHashGetDigestSize(hashAlg);

// if this is a signing or a decryption key, then the limit
// for the data size is the block size of the hash. This limit
// is set because larger values have lower entropy because of the
// HMAC function. The lower limit is 1/2 the size of the digest
//
// If the user provided the key, check that it is a proper size
if (sensitiveCreate->data.t.size != 0)
{
  if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, decrypt)
      || IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
  {
    if (sensitiveCreate->data.t.size > hashBlockSize)
      TPM_RC_SIZE;
    #if 0
      // May make this a FIPS-mode requirement
      if (sensitiveCreate->data.t.size < (digestSize / 2))
        return TPM_RC_SIZE;
    #endif
  }
  // If this is a data blob, then anything that will get past the unmarshaling
  // is OK
  MemoryCopy2B(&sensitive->sensitive.bits.b, &sensitiveCreate->data.b,
                sizeof(sensitive->sensitive.bits.t.buffer));
  }
else
{
  // The TPM is going to generate the data so set the size to be the
  // size of the digest of the algorithm
  sensitive->sensitive.bits.t.size =
      DRBG_Generate(rand, sensitive->sensitive.bits.t.buffer, digestSize);
  if (sensitive->sensitive.bits.t.size == 0)
    return TPM_RC_NO_RESULT;
  }
  return TPM_RC_SUCCESS;
}

10.2.6.3.4 CryptIsSchemeAnonymous()

This function is used to test a scheme to see if it is an anonymous scheme. The only anonymous scheme is ECDAA. ECDAA can be used to do things like U-Prove.

BOOL CryptIsSchemeAnonymous(
    TPM_ALG_ID scheme // IN: the scheme algorithm to test
)
{
  return scheme == ALG_ECDAA_VALUE;
}

10.2.6.4 Symmetric Functions

10.2.6.4.1 ParmDecryptSym()

This function performs parameter decryption using symmetric block cipher.
128  TPM_ALG_ID   symAlg,    // IN: the symmetric algorithm
129  TPM_ALG_ID   hash,      // IN: hash algorithm for KDFa
130  UINT16       keySizeInBits,  // IN: the key size in bits
131  TPM2B        *key,       // IN: KDF HMAC key
132  TPM2B        *nonceCaller,  // IN: nonce caller
133  TPM2B        *nonceTpm,   // IN: nonce TPM
134  UINT32       dataSize,   // IN: size of parameter buffer
135  BYTE         *data       // OUT: buffer to be decrypted
136 )
137 {
138    // KDF output buffer
139    // It contains parameters for the CFB encryption
140    // From MSB to LSB, they are the key and iv
141    BYTE symParmString[MAX_SYM_KEY_BYTES + MAX_SYM_BLOCK_SIZE];
142    // Symmetric key size in byte
143    UINT16 keySize = (keySizeInBits + 7) / 8;
144    TPM2B_IV iv;
145
146    iv.t.size = CryptGetSymmetricBlockSize(symAlg, keySizeInBits);
147    if(iv.t.size > 0)
148    {
149      // Generate key and iv
150      CryptKDFa(hash, key, CFB_KEY, nonceCaller, nonceTpm,
151                  keySizeInBits + (iv.t.size * 8), symParmString, NULL, FALSE);
152      MemoryCopy(iv.t.buffer, &symParmString[keySize], iv.t.size);
153      CryptSymmetricDecrypt(data, symAlg, keySizeInBits, symParmString,
154                                &iv, ALG_CFB_VALUE, dataSize, data);
155      return;
156    }
157 // End ParmEncryptSym()
158
10.2.6.4.2 ParmEncryptSym()

This function performs parameter encryption using symmetric block cipher.
keySizeInBits + (iv.t.size * 8), symParmString, NULL, FALSE);
MemoryCopy(iv.t.buffer, &symParmString[keySize], iv.t.size);
CryptSymmetricEncrypt(data, symAlg, keySizeInBits, symParmString, &iv,
ALG_CFB_VALUE, dataSize, data);
return;
}

10.2.6.4.3 CryptGenerateKeySymmetric()

This function generates a symmetric cipher key. The derivation process is determined by the type of the provided rand

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>cannot get a random value</td>
</tr>
<tr>
<td>TPM_RC_KEY_SIZE</td>
<td>key size in the public area does not match the size in the sensitive creation area</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>provided key value is not allowed</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
CryptGenerateKeySymmetric(
    TMPT_PUBLIC *publicArea, // IN/OUT: The public area template
    // IN/OUT: The public area template
    TMPT_SENSITIVE *sensitive, // OUT: sensitive area
    TPMS_SENSITIVE_CREATE *sensitiveCreate, // IN: sensitive creation data
    RAND_STATE *rand // IN: the "entropy" source for
)
{
    UINT16 keyBits = publicArea->parameters.symDetail.sym.keyBits.sym;
    TPM_RC result;
    // only do multiples of RADIX_BITS
    if((keyBits % RADIX_BITS) != 0)
        return TPM_RC_KEY_SIZE;
    // If this is not a new key, then the provided key data must be the right size
    if(sensitiveCreate->data.t.size != 0)
    {
        result = CryptSymKeyValidate(&publicArea->parameters.symDetail.sym,
        (TPM2B_SYM_KEY *)sensitiveCreate->data);
        if(result == TPM_RC_SUCCESS)
            MemoryCopy2B(sensitive->sym.b, sensitiveCreate->data.b,
            sizeof(sensitive->sym.t.buffer));
    }
    #if ALG_TDES
    else if(publicArea->parameters.symDetail.sym.algorithm == ALG_TDES_VALUE)
    {
        result = CryptGenerateKeyDes(publicArea, sensitive, rand);
    }
    #endif
    else
    {
        sensitive->sym.t.size =
        DRBG_Generate(rand, sensitive->sym.t.buffer,
        BITS_TO_BYTES(keyBits));
        result = (sensitive->sym.t.size == 0)
        ? TPM_RC_NO_RESULT : TPM_RC_SUCCESS;
    }
    return result;
}
10.2.6.4 CryptXORObfuscation()

This function implements XOR obfuscation. It should not be called if the hash algorithm is not implemented. The only return value from this function is TPM_RC_SUCCESS.

```c
void CryptXORObfuscation(
    TPM_ALG_ID       hash,  // IN: hash algorithm for KDF
    TPM2B           *key,   // IN: KDF key
    TPM2B           *contextU, // IN: contextU
    TPM2B           *contextV, // IN: contextV
    UINT32           dataSize, // IN: size of data buffer
    BYTE            *data    // IN/OUT: data to be XORed in place
)
{
    BYTE             mask[MAX_DIGEST_SIZE];  // Allocate a digest sized buffer
    BYTE            *pm;
    UINT32           i;
    UINT32           counter = 0;
    UINT16           hLen = CryptHashGetDigestSize(hash);
    UINT32           requestSize = dataSize * 8;
    INT32            remainBytes = (INT32)dataSize;

    pAssert((key != NULL) && (data != NULL) && (hLen != 0));

    // Call KDFa to generate XOR mask
    for(; remainBytes > 0; remainBytes -= hLen)
    {
        // Make a call to KDFa to get next iteration
        CryptKDFa(hash, key, XOR_KEY, contextU, contextV,
                  requestSize, mask, &counter, TRUE);

        // XOR next piece of the data
        pm = mask;
        for(i = hLen < remainBytes ? hLen : remainBytes; i > 0; i--)
            *data++ ^= *pm++;
    }
    return;
}
```

10.2.6.5 Initialization and shut down

10.2.6.5.1 CryptInit()

This function is called when the TPM receives a _TPM_Init() indication.

**NOTE:** The hash algorithms do not have to be tested, they just need to be available. They have to be tested before the TPM can accept HMAC authorization or return any result that relies on a hash algorithm.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>initializations succeeded</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>initialization failed and caller should place the TPM into Failure Mode</td>
</tr>
</tbody>
</table>

```c
BOOL CryptInit(
    void
)
{
    BOOL ok;
    // Initialize the vector of implemented algorithms
```
AlgorithmGetImplementedVector(\&g\_implementedAlgorithms);

// Indicate that all test are necessary
CryptInitializeToTest();

// Do any library initializations that are necessary. If any fails,
// the caller should go into failure mode;
ok = SupportLibInit();
ok = ok && CryptSymInit();
ok = ok && CryptRandInit();
ok = ok && CryptHashInit();
#if ALG_RSA
ok = ok && CryptRsaInit();
#endif // ALG_RSA
#if ALG_ECC
ok = ok && CryptEccInit();
#endif // ALG_ECC
return ok;

10.2.6.5.2 CryptStartup()

This function is called by TPM2_Startup() to initialize the functions in this cryptographic library and in the
provided CryptoLibrary(). This function and CryptUtilInit() are both provided so that the implementation
may move the initialization around to get the best interaction.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>startup succeeded</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>startup failed and caller should place the TPM into Failure Mode</td>
</tr>
</tbody>
</table>

BOOL CryptStartup(
    STARTUP_TYPE     type    // IN: the startup type
)
{
    BOOL            OK;
    NOT_REFERENCED(type);
    OK = CryptSymStartup() && CryptRandStartup() && CryptHashStartup()
#if ALG_RSA
    && CryptRsaStartup()
#endif // ALG_RSA
#if ALG_ECC
    && CryptEccStartup()
#endif // ALG_ECC
    ;
#if ALG_ECC
    // Don't directly check for SU_RESET because that is the default
    if(OK && (type != SU_RESTART) && (type != SU_RESUME))
    {
        // If the shutdown was orderly, then the values recovered from NV will
        // be OK to use.
        // Get a new random commit nonce
        gr.commitNonce.t.size = sizeof(gr.commitNonce.t.buffer);
        CryptRandomGenerate(gr.commitNonce.t.size, gr.commitNonce.t.buffer);
        // Reset the counter and commit array
        gr.commitCounter = 0;
        MemorySet(gr.commitArray, 0, sizeof(gr.commitArray));
    }
#endif // ALG_ECC
    return OK;
}
10.2.6.6 Algorithm-Independent Functions

10.2.6.6.1 Introduction

These functions are used generically when a function of a general type (e.g., symmetric encryption) is required. The functions will modify the parameters as required to interface to the indicated algorithms.

10.2.6.6.2 CryptIsAsymAlgorithm()

This function indicates if an algorithm is an asymmetric algorithm.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>if it is an asymmetric algorithm</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>if it is not an asymmetric algorithm</td>
</tr>
</tbody>
</table>

```c
BOOL CryptIsAsymAlgorithm(
    TPM_ALG_ID       algID       // IN: algorithm ID
) {
    switch(algID) {
        #if ALG_RSA
            case ALG_RSA_VALUE:
                #endif
                return TRUE;
        #if ALG_ECC
            case ALG_ECC_VALUE:
                #endif
                return TRUE;
        default:
            break;
    }
    return FALSE;
}
```

10.2.6.6.3 CryptSecretEncrypt()

This function creates a secret value and its associated secret structure using an asymmetric algorithm. This function is used by TPM2_Rewrap(), TPM2_MakeCredential(), and TPM2_Duplicate().

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>keyHandle does not reference a valid decryption key</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>invalid ECC key (public point is not on the curve)</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>RSA key with an unsupported padding scheme</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>numeric value of the data to be decrypted is greater than the RSA key modulus</td>
</tr>
</tbody>
</table>

```c
TPM_RC
CryptSecretEncrypt(
    OBJECT                  *encryptKey,  // IN: encryption key object
    const TPM2B             *label,      // IN: a null-terminated string as L
    TPM2B_DATA              *data,       // OUT: secret value
    TPM2B_ENCRYPTED_SECRET  *secret     // OUT: secret structure
)
```
```c
354 {
355   TPM2B_PUBLIC_KEY_RSA *secret = (TPM2B_PUBLIC_KEY_RSA *)secret;
356   result = TPM_RC_SUCCESS;
357   //
358   if(data == NULL || secret == NULL)
359     return TPM_RC_FAILURE;
360   // The output secret value has the size of the digest produced by the nameAlg.
361   data->t.size = CryptHashGetDigestSize(encryptKey->publicArea.nameAlg);
362   // The encryption scheme is OAEP using the nameAlg of the encrypt key.
363   scheme.scheme = ALG_OAEP_VALUE;
364   scheme.details.anySig.hashAlg = encryptKey->publicArea.nameAlg;
365   if(!IS_ATTRIBUTE(encryptKey->publicArea.objectAttributes, TPMA_OBJECT, decrypt))
366     return TPM_RC_ATTRIBUTES;
367   switch(encryptKey->publicArea.type)
368   {
369     #if ALG_RSA
370     case ALG_RSA_VALUE:
371       { // Create secret data from RNG
372         CryptRandomGenerate(data->t.size, data->t.buffer);
373         // Encrypt the data by RSA OAEP into encrypted secret
374         result = CryptRsaEncrypt((TPM2_PUBLIC_KEY_RSA *)secret, &data->b,
375                               encryptKey, &scheme, label, NULL);
376       }
377     break;
378     #endif // ALG_RSA
379     #if ALG_ECC
380     case ALG_ECC_VALUE:
381       { TPMS_ECC_POINT eccPublic;
382         TPMS_ECC_POINT eccSecret;
383         BYTE *buffer = secret->t.secret;
384         // Need to make sure that the public point of the key is on the
385         // curve defined by the key.
386         if(!CryptEccIsPointOnCurve(encryptKey->publicArea.parameters.eccDetail.curveID,
387                                     &encryptKey->publicArea.unique.ecc))
388           result = TPM_RC_KEY;
389         else
390           { // Call crypto engine to create an auxiliary ECC key
391             // We assume crypt engine initialization should always success.
392             // Otherwise, TPM should go to failure mode.
393             CryptEccNewKeyPair(&eccPublic, &eccPrivate,
394                                encryptKey->publicArea.parameters.eccDetail.curveID);
395             // Marshal ECC public to secret structure. This will be used by the
396             // recipient to decrypt the secret with their private key.
397             secret->t.size = TPMS_ECC_POINT_Marshal(&eccPublic, &buffer, NULL);
398             // Compute ECDH shared secret which is R = [d]Q where d is the
399             // private part of the ephemeral key and Q is the public part of a
400             // TPM key. TPM_RC_KEY error return from CryptComputeECDHSecret
401             // because the auxiliary ECC key is just created according to the
402             // parameters of input ECC encrypt key.
403             if(CryptEccPointMultiply(&eccSecret, encryptKey->publicArea.parameters.eccDetail.curveID,
404                                         &encryptKey->publicArea.unique.ecc, &eccPrivate,
405                                         NULL, NULL)
406               != TPM_RC_SUCCESS)
407               return TPM_RC_FAILURE;
408           }
409         }
410       }
411     break;
412     #endif // ALG_ECC
413   }
414   }
```
result = TPM_RC_KEY;
else
{
    // The secret value is computed from Z using KDFe as:
    // secret := KDFe(HashID, Z, Use, PartyUInfo, PartyVInfo, bits)
    // Where:
    // HashID  the nameAlg of the decrypt key
    // Z   the x coordinate (Px) of the product (P) of the point
    // (Q) of the secret and the private x coordinate (de,V)
    // of the decryption key
    // Use a null-terminated string containing "SECRET"
    // PartyUInfo the x coordinate of the point in the secret
    // (Qe,U )
    // PartyVInfo the x coordinate of the public key (Qs,V )
    // bits the number of bits in the digest of HashID
    // Retrieve seed from KDFe
    CryptKDFe(encryptKey->publicArea.nameAlg, &eccSecret.x.b,
              label, &eccPublic.x.b,
              &encryptKey->publicArea.unique.ecc.x.b,
              data->t.size * 8, data->t.buffer);
}
break;
#endif // ALG_ECC
default:
    FAIL(FATAL_ERROR_INTERNAL);
    break;
}
return result;

10.2.6.6.4 CryptSecretDecrypt()

Decrypt a secret value by asymmetric (or symmetric) algorithm This function is used for
ActivateCredential() and Import for asymmetric decryption, and StartAuthSession() for both asymmetric
and symmetric decryption process

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>RSA key is not a decryption key</td>
</tr>
<tr>
<td>TPM_RC_BINDING</td>
<td>Invalid RSA key (public and private parts are not cryptographically bound.</td>
</tr>
<tr>
<td>TPM_RC_ECC_POINT</td>
<td>ECC point in the secret is not on the curve</td>
</tr>
<tr>
<td>TPM_RC_INSUFFICIENT</td>
<td>failed to retrieve ECC point from the secret</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>multiplication resulted in ECC point at infinity</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>data to decrypt is not of the same size as RSA key</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>For RSA key, numeric value of the encrypted data is greater than the modulus, or the recovered data is larger than the output buffer. For keyedHash or symmetric key, the secret is larger than the size of the digest produced by the name algorithm.</td>
</tr>
<tr>
<td>TPM_RC_FAILURE</td>
<td>internal error</td>
</tr>
</tbody>
</table>

451
452
453 CryptSecretDecrypt(
454    OBJECT *decryptKey,  // IN: decrypt key
455    TPM2B_NONCE *nonceCaller,  // IN: nonceCaller. It is needed for
456    // symmetric decryption. For
457    // asymmetric decryption, this
458    void
459    )
457        const TPM2B             *label,   // parameter is NULL
458        TPM2B_ENCRYPTED_SECRET *secret,   // IN: a value for L
459        TPM2B_DATA              *data     // IN: input secret
460        TPM2B_DATA              *data     // OUT: decrypted secret value
461    }
462
463    TPM_RC      result = TPM_RC_SUCCESS;
464
465    // Decryption for secret
466    switch(decryptKey->publicArea.type)
467    {
468        if        ALG_RSA
469            case ALG_RSA_VALUE:
470                {  
471                    // Decryption for secret
472                    // Decryption for secret
473                    switch(decryptKey->publicArea.type)
474                    {  
475                        #if ALG_RSA
476                        case ALG_RSA_VALUE:
477                            {  
478                                // Decryption for secret
479                                // Decryption for secret
480                                switch(decryptKey->publicArea.type)
481                                {  
482                                    // Decryption for secret
483                                    // Decryption for secret
484                                    switch(decryptKey->publicArea.type)
485                                    {  
486                                        // Decryption for secret
487                                        // Decryption for secret
488                                        switch(decryptKey->publicArea.type)
489                                        {  
490                                            // Decryption for secret
491                                            // Decryption for secret
492                                            switch(decryptKey->publicArea.type)
493                                            {  
494                                                // Decryption for secret
495                                                // Decryption for secret
496                                                switch(decryptKey->publicArea.type)
497                                                {  
498                                                    // Decryption for secret
499                                                    // Decryption for secret
500                                                    switch(decryptKey->publicArea.type)
501                                                    {  
502                                                        // Decryption for secret
503                                                        // Decryption for secret
504                                                        switch(decryptKey->publicArea.type)
505                                                        {  
506                                                            // Decryption for secret
507                                                            // Decryption for secret
508                                                            switch(decryptKey->publicArea.type)
509                                                            {  
510                                                                // Decryption for secret
511                                                                // Decryption for secret
512                                                                switch(decryptKey->publicArea.type)
513                                                                {  
514                                                                    // Decryption for secret
515                                                                    // Decryption for secret
516                                                                    switch(decryptKey->publicArea.type)
517                                                                    {  
518                                                                        // Decryption for secret
519                                                                        // Decryption for secret
520                                                                        switch(decryptKey->publicArea.type)
521                                                                        {  
522                                                                            // Decryption for secret
523                                                                            // Decryption for secret
524                                                                            switch(decryptKey->publicArea.type)
525                                                                            {  
526                                                                                // Decryption for secret
527                                                                                // Decryption for secret
528                                                                                switch(decryptKey->publicArea.type)
529                                                                                {  
530                                                                                    // Decryption for secret
531                                                                                    // Decryption for secret
532                                                                                    switch(decryptKey->publicArea.type)
533                                                                                    {  
534                                                                做出贡献。
if (result == TPM_RC_SUCCESS) {
    // Set the size of the "recovered" secret value to be the size
    // of the digest produced by the nameAlg.
    data->t.size = 
    CryptHashGetDigestSize(decryptKey->publicArea.nameAlg);

    // The secret value is computed from Z using KDFe as:
    // secret := KDFe(HashID, Z, Use, PartyUInfo, PartyVInfo, bits)
    // Where:
    // HashID -- the nameAlg of the decrypt key
    // Z -- the x coordinate (Px) of the product (P) of the point
    // (Q) of the secret and the private x coordinate (de,V)
    // of the decryption key
    // Use -- a null-terminated string containing "SECRET"
    // PartyUInfo -- the x coordinate of the point in the secret
    // PartyVInfo -- the x coordinate of the public key (Qs,V)
    // bits -- the number of bits in the digest of HashID
    // Retrieve seed from KDFe
    CryptKDFe(decryptKey->publicArea.nameAlg, &eccSecret.x.b, label,
    &eccPublic.x.b, &decryptKey->publicArea.unique.ecc.x.b, 
    data->t.size * 8, data->t.buffer);
}

if (secret->t.size > 
    CryptHashGetDigestSize(decryptKey->publicArea.nameAlg))
    result = TPM_RC_VALUE;
else
    {
        // Retrieve seed by XOR Obfuscation:
        //    seed = XOR(secret, hash, key, nonceCaller, nullNonce)
        // where:
        //    secret the secret parameter from the TPM2_StartAuthHMAC
        //    command that contains the seed value
        //    hash nameAlg of tpmKey
        //    key the key or data value in the object referenced by
        //    entityHandle in the TPM2_StartAuthHMAC command
        //    nonceCaller the parameter from the TPM2_StartAuthHMAC command
        //    nullNonce a zero-length nonce
        // XOR Obfuscation in place
        CryptXORObfuscation(decryptKey->publicArea.nameAlg, 
        &decryptKey->sensitive.sensitive.bits.b, 
        &nonceCaller->b, NULL, 
        secret->t.size, secret->t.secret);

        // Copy decrypted seed
        MemoryCopy2B(&data->b, &secret->b, sizeof(data->t.buffer));
    }

    break;
}

case ALG_KEYEDHASH_VALUE:
    
    // The seed size can not be bigger than the digest size of nameAlg
    if (secret->t.size > 
        CryptHashGetDigestSize(decryptKey->publicArea.nameAlg))
        result = TPM_RC_VALUE;
    else
        {
            // Retrieve seed by XOR Obfuscation:
            //    seed = XOR(secret, hash, key, nonceCaller, nullNonce)
            // where:
            //    secret the secret parameter from the TPM2_StartAuthHMAC
            //    command that contains the seed value
            //    hash nameAlg of tpmKey
            //    key the key or data value in the object referenced by
            //    entityHandle in the TPM2_StartAuthHMAC command
            //    nonceCaller the parameter from the TPM2_StartAuthHMAC command
            //    nullNonce a zero-length nonce
            // XOR Obfuscation in place
            CryptXORObfuscation(decryptKey->publicArea.nameAlg, 
            &decryptKey->sensitive.sensitive.bits.b, 
            &nonceCaller->b, NULL, 
            secret->t.size, secret->t.secret);

            // Copy decrypted seed
            MemoryCopy2B(&data->b, &secret->b, sizeof(data->t.buffer));
        }
    break;
}

case ALG_SYMCIPHER_VALUE:
    
    TPM2B_IV                iv = {{0}}; 
    TPMT_SYM_DEF_OBJECT     *symDef; 
    
    // The seed size can not be bigger than the digest size of nameAlg
    if (secret->t.size > 
        CryptHashGetDigestSize(decryptKey->publicArea.nameAlg))
        result = TPM_RC_VALUE;
    else
        {
            // Retrieve seed by XOR Obfuscation:
            //    seed = XOR(secret, hash, key, nonceCaller, nullNonce)
            // where:
            //    secret the secret parameter from the TPM2_StartAuthHMAC
            //    command that contains the seed value
            //    hash nameAlg of tpmKey
            //    key the key or data value in the object referenced by
            //    entityHandle in the TPM2_StartAuthHMAC command
            //    nonceCaller the parameter from the TPM2_StartAuthHMAC command
            //    nullNonce a zero-length nonce
            // XOR Obfuscation in place
            CryptXORObfuscation(decryptKey->publicArea.nameAlg, 
            &decryptKey->sensitive.sensitive.bits.b, 
            &nonceCaller->b, NULL, 
            secret->t.size, secret->t.secret);

            // Copy decrypted seed
            MemoryCopy2B(&data->b, &secret->b, sizeof(data->t.buffer));
        }
    break;
return TPM_RC_VALUE;

else {
    symDef = &decryptKey->publicArea.parameters.symDetail.sym;
    iv.t.size = CryptGetSymmetricBlockSize(symDef->algorithm,
                                           symDef->keyBits.sym);
    if (iv.t.size == 0)
        return TPM_RC_FAILURE;
    if (nonceCaller->t.size >= iv.t.size)
        MemoryCopy(iv.t.buffer, nonceCaller->t.buffer, iv.t.size);
    else {
        if (nonceCaller->t.size > sizeof(iv.t.buffer))
            return TPM_RC_FAILURE;
        MemoryCopy(iv.b.buffer, nonceCaller->t.buffer,
                   nonceCaller->t.size);
    }
    // make sure secret will fit
    if (secret->t.size > data->t.size)
        return TPM_RC_FAILURE;
    data->t.size = secret->t.size;
    // CFB decrypt, using nonceCaller as iv
    CryptSymmetricDecrypt(data->t.buffer, symDef->algorithm,
                           symDef->keyBits.sym,
                           decryptKey->sensitive.sensitive.sym.t.buffer,
                           &iv, ALG_CFB_VALUE, secret->t.size,
                           secret->t.secret);
}
}
break;
default:
    FAIL(FATAL_ERROR_INTERNAL);
break;
}
return result;
}

10.2.6.6.5 CryptParameterEncryption()

This function does in-place encryption of a response parameter.

void CryptParameterEncryption(
    TPM_HANDLE handle,       // IN: encrypt session handle
    TPM2B *nonceCaller,      // IN: nonce caller
    UINT16 leadingSizeInByte, // IN: the size of the leading size field in bytes
    TPM2B_AUTH *extraKey,    // IN: additional key material other than sessionAuth
    BYTE *buffer              // IN/OUT: parameter buffer to be encrypted
)
{
    SESSION *session = SessionGet(handle); // encrypt session
    TPM2B_TYPE(TEMP_KEY, (sizeof(extraKey->t.buffer)
            + sizeof(session->sessionKey.t.buffer)));
    TPM2B_TEMP_KEY key; // encryption key
    UINT32 cipherSize = 0; // size of cipher text
    // Retrieve encrypted data size.
    if (leadingSizeInByte == 2)
        // Extract the first two bytes as the size field as the data size
// encrypt
cipherSize = (UINT32)BYTE_ARRAY_TO_UINT16(buffer);
// advance the buffer
buffer = &buffer[2];
}
#endif

else if(leadingSizeInByte == 4)
{
    // use the first four bytes to indicate the number of bytes to encrypt
cipherSize = BYTE_ARRAY_TO_UINT32(buffer);
    // advance pointer
    buffer = &buffer[4];
}
#endif
else
{
    FAIL(FATAL_ERROR_INTERNAL);
}

// Compute encryption key by concatenating sessionKey with extra key
MemoryCopy2B(&key.b, &session->sessionKey.b, sizeof(key.t.buffer));
MemoryConcat2B(&key.b, &extraKey->b, sizeof(key.t.buffer));

if(session->symmetric.algorithm == ALG_XOR_VALUE)
{
    // XOR parameter encryption formulation:
    //    XOR(parameter, hash, sessionAuth, nonceNewer, nonceOlder)
    CryptXORObfuscation(session->authHashAlg, &(key.b),
                 &(session->nonceTPM.b),
                 nonceCaller, cipherSize, buffer);
    else
        ParmEncryptSym(session->symmetric.algorithm, session->authHashAlg,
                        session->symmetric.keyBits.aes, &(key.b),
                        nonceCaller, &(session->nonceTPM.b),
                        cipherSize, buffer);
    return;
}

10.2.6.6.6 CryptParameterDecryption()

This function does in-place decryption of a command parameter.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIZE</td>
<td>The number of bytes in the input buffer is less than the number of bytes to be decrypted.</td>
</tr>
</tbody>
</table>

TPM_RC

CryptParameterDecryption(

  TPM_HANDLE       handle,  // IN: encrypted session handle
  TPM2B           *nonceCaller,  // IN: nonce caller
  UINT32           bufferSize,  // IN: size of parameter buffer
  UINT16           leadingSizeInByte,  // IN: the size of the leading size field in byte
  TPM2B_AUTH      *extraKey,  // IN: the authValue
  BYTE            *buffer  // IN/OUT: parameter buffer to be decrypted
)
{
    SESSION         *session = SessionGet(handle);  // encrypt session
    // The HMAC key is going to be the concatenation of the session key and any
    // additional key material (like the authValue). The size of both of these
    // is the size of the buffer which can contain a TPMT_HA.
    TPM2B_TYPE(HMAC_KEY, (sizeof(extraKey->t.buffer)
                     + sizeof(session->sessionKey.t.buffer)));
702    TPM2B_HMAC_KEY          key;       // decryption key
703    UINT32                  cipherSize = 0; // size of cipher text
704
705    // Retrieve encrypted data size.
706    if(leadingSizeInByte == 2)
707    {
708        // The first two bytes of the buffer are the size of the
709        // data to be decrypted
710        cipherSize = (UINT32)BYTE_ARRAY_TO_UINT16(buffer);
711        buffer = &buffer[2]; // advance the buffer
712    }
713    #ifdef TPM4B
714    else if(leadingSizeInByte == 4)
715    {
716        // the leading size is four bytes so get the four byte size field
717        cipherSize = BYTE_ARRAY_TO_UINT32(buffer);
718        buffer = &buffer[4]; // advance pointer
719    }
720    #endif
721    else
722    {
723        FAIL(FATAL_ERROR_INTERNAL);
724    }
725    if(cipherSize > bufferSize)
726        return TPM_RC_SIZE;
727
728    // Compute decryption key by concatenating sessionAuth with extra input key
729    MemoryCopy2B(&key.b, &session->sessionKey.b, sizeof(key.t.buffer));
730    MemoryConcat2B(&key.b, &extraKey->b, sizeof(key.t.buffer));
731
732    if(session->symmetric.algorithm == ALG_XOR_VALUE)
733    { // XOR parameter decryption formulation:
734        // XOR(parameter, hash, sessionAuth, nonceNewer, nonceOlder)
735        // Call XOR obfuscation function
736        CryptXORObfuscation(session->authHashAlg, &key.b, nonceCaller,
737                        &session->nonceTPM.b, cipherSize, buffer);
738    } else
739    { // Assume that it is one of the symmetric block ciphers.
740        ParmDecryptSym(session->symmetric.algorithm, session->authHashAlg,
741                        session->symmetric.keyBits.sym,
742                        &key.b, nonceCaller, &session->nonceTPM.b,
743                        cipherSize, buffer);
744    }
745    return TPM_RC_SUCCESS;
746
10.2.6.6.7 CryptComputeSymmetricUnique()

This function computes the unique field in public area for symmetric objects.

void
CryptComputeSymmetricUnique()

TPMT_PUBLIC     *publicArea,  // IN: the object's public area
TPMT_SENSITIVE  *sensitive,  // IN: the associated sensitive area
TPM2B_DIGEST    *unique    // OUT: unique buffer

{ // For parents (symmetric and derivation), use an HMAC to compute
    // the 'unique' field
    if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted)
        && IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, decrypt))
    {
        // Unique field is HMAC(sensitive->seedValue, sensitive->sensitive)
        HMAC_STATE hmacState;
        }
10.2.6.6.8 CryptCreateObject()

This function creates an object. For an asymmetric key, it will create a key pair and, for a parent key, a seed value for child protections.

For a symmetric object, (TPM_ALG_SYMCIPHER or TPM_ALG_KEYEDHASH), it will create a secret key if the caller did not provide one. It will create a random secret seed value that is hashed with the secret value to create the public unique value.

publicArea, sensitive, and sensitiveCreate are the only required parameters and are the only ones that are used by TPM2_Create(). The other parameters are optional and are used when the generated Object needs to be deterministic. This is the case for both Primary Objects and Derived Objects.

When a seed value is provided, a RAND_STATE will be populated and used for all operations in the object generation that require a random number. In the simplest case, TPM2_CreatePrimary() will use seed, label and context with context being the hash of the template. If the Primary Object is in the Endorsement hierarchy, it will also populate proof with ehProof.

For derived keys, seed will be the secret value from the parent, label and context will be set according to the parameters of TPM2_CreateLoaded() and hashAlg will be set which causes the RAND_STATE to be a KDF generator.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>a provided key is not an allowed value</td>
</tr>
<tr>
<td>TPM_RC_KEY_SIZE</td>
<td>key size in the public area does not match the size in the sensitive creation area for a symmetric key</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>unable to get random values (only in derivation)</td>
</tr>
<tr>
<td>TPM_RC_RANGE</td>
<td>for an RSA key, the exponent is not supported</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>sensitive data size is larger than allowed for the scheme for a keyed hash object</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>exponent is not prime or could not find a prime using the provided parameters for an RSA key; unsupported name algorithm for an ECC key</td>
</tr>
</tbody>
</table>

```c
TPM_RC
CryptCreateObject(
    OBJECT                  *object,                // IN: new object structure pointer
    TPMS_SENSITIVE_CREATE   *sensitiveCreate,      // IN: sensitive creation
    RAND_STATE              *rand                   // IN: the random number generator
)
```

10.2.6.6.8 CryptCreateObject()

This function creates an object. For an asymmetric key, it will create a key pair and, for a parent key, a seed value for child protections.

For a symmetric object, (TPM_ALG_SYMCIPHER or TPM_ALG_KEYEDHASH), it will create a secret key if the caller did not provide one. It will create a random secret seed value that is hashed with the secret value to create the public unique value.

publicArea, sensitive, and sensitiveCreate are the only required parameters and are the only ones that are used by TPM2_Create(). The other parameters are optional and are used when the generated Object needs to be deterministic. This is the case for both PrimaryObjects and Derived Objects.

When a seed value is provided, a RAND_STATE will be populated and used for all operations in the object generation that require a random number. In the simplest case, TPM2_CreatePrimary() will use seed, label and context with context being the hash of the template. If the Primary Object is in the Endorsement hierarchy, it will also populate proof with ehProof.

For derived keys, seed will be the secret value from the parent, label and context will be set according to the parameters of TPM2_CreateLoaded() and hashAlg will be set which causes the RAND_STATE to be a KDF generator.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>a provided key is not an allowed value</td>
</tr>
<tr>
<td>TPM_RC_KEY_SIZE</td>
<td>key size in the public area does not match the size in the sensitive creation area for a symmetric key</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>unable to get random values (only in derivation)</td>
</tr>
<tr>
<td>TPM_RC_RANGE</td>
<td>for an RSA key, the exponent is not supported</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>sensitive data size is larger than allowed for the scheme for a keyed hash object</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>exponent is not prime or could not find a prime using the provided parameters for an RSA key; unsupported name algorithm for an ECC key</td>
</tr>
</tbody>
</table>

```c
TPM_RC
CryptCreateObject(
    OBJECT                  *object,                // IN: new object structure pointer
    TPMS_SENSITIVE_CREATE   *sensitiveCreate,      // IN: sensitive creation
    RAND_STATE              *rand                   // IN: the random number generator
)
```
{
    TPM_PUBLIC   *publicArea = &object->publicArea;
    TPM_RC       result = TPM_RC_SUCCESS;

    // Set the sensitive type for the object
    object->sensitive.sensitiveType = publicArea->type;

    // For all objects, copy the initial authorization data
    object->sensitive.authValue = sensitiveCreate->userAuth;

    // If the TPM is the source of the data, set the size of the provided data to
    // zero so that there's no confusion about what to do.
    if(IS_ATTRIBUTE(object->publicArea.objectAttributes,
                    TPMA_OBJECT, sensitiveDataOrigin))
        sensitiveCreate->data.t.size = 0;

    // Generate the key and unique fields for the asymmetric keys and just the
    // sensitive value for symmetric object
    switch(publicArea->type)
    {
    #if ALG_RSA
     // Create RSA key
     case ALG_RSA_VALUE:
      // RSA uses full object so that it has a place to put the private
      // exponent
      result = CryptRsaGenerateKey(object, rand);
      break;
    #endif // ALG_RSA

    #if ALG_ECC
    // Create ECC key
    case ALG_ECC_VALUE:
    result = CryptEccGenerateKey(&object->publicArea, &object->sensitive,
                                  rand);
    break;
    #endif // ALG_ECC

    case ALG_SYMCIPHER_VALUE:
    result = CryptGenerateKeySymmetric(&object->publicArea,
                                       &object->sensitive,
                                       sensitiveCreate, rand);
    break;

    case ALG_KEYEDHASH_VALUE:
    result = CryptGenerateKeyedHash(&object->publicArea,
                                    &object->sensitive,
                                    sensitiveCreate, rand);
    break;

    default:
    FAIL(FATAL_ERROR_INTERNAL);
    break;
    }

    if(result != TPM_RC_SUCCESS)
        return result;

    // Create the sensitive seed value
    // If this is a primary key in the endorsement hierarchy, stir the DRBG state
    // This implementation uses both shProof and ehProof to make sure that there
    // is no leakage of either.
    if(object->attributes.primary && object->attributes.epsHierarchy)
    {
        DRBG_AdditionalData((DRBG_STATE *)rand, &gp.shProof.b);
        DRBG_AdditionalData((DRBG_STATE *)rand, &gp.ehProof.b);
    }

    // Generate a seedValue that is the size of the digest produced by nameAlg
    object->sensitive.seedValue.t.size =
        DRBG_Generate(rand, object->sensitive.seedValue.t.buffer,
                      CryptHashGetDigestSize(publicArea->nameAlg));

    if(object->sensitive.seedValue.t.size == 0)
        return TPM_RC_NO_RESULT;
// For symmetric objects, need to compute the unique value for the public area
852 if (publicArea->type == ALG_SYMCIPHER_VALUE
853 || publicArea->type == ALG_KEYEDHASH_VALUE)
854 {
855    CryptComputeSymmetricUnique(&object->publicArea, &object->sensitive,
856                                &object->publicArea.unique.sym);
857 }
858 else
859 {
860    // if this is an asymmetric key and it isn't a parent, then
861    // get rid of the seed.
862    if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign)
863       || !IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted))
864        memset(&object->sensitive.seedValue, 0,
865               sizeof(object->sensitive.seedValue));
866 }
867 // Compute the name
868 PublicMarshalAndComputeName(&object->publicArea, &object->name);
869 return result;
870 }

10.2.6.6.9 CryptGetSignHashAlg()

Get the hash algorithm of signature from a TPMT_SIGNATURE structure. It assumes the signature is not NULL. This is a function for easy access.

871 TPMI_ALG_HASH
872 CryptGetSignHashAlg(
873    TPMT_SIGNATURE *auth  // IN: signature
874 )
875 {
876     if (auth->sigAlg == ALG_NULL_VALUE)
877         FAIL(FATAL_ERROR_INTERNAL);
878     switch (auth->sigAlg)
879     {
880         #if ALG_RSA
881             // If RSA is supported, both RSASSA and RSAPSS are required
882         #if !defined ALG_RSASSA_VALUE || !defined ALG_RSAPSS_VALUE
883             #error "RSASSA and RSAPSS are required for RSA"
884         #endif
885         #endif // ALG_RSA
886         #if ALG_ECC
887             // If ECC is defined, ECDSA is mandatory
888         #if !ALG_ECDSA
889             #error "ECDSA is required for ECC"
890         #endif
891         #endif // ALG_ECC
892         #if ALG_SM2
893             case ALG_SM2_VALUE:
894                 return auth->signature.rsassa.hash;
895         #endif
896         #if ALG_ECSCHNORR
897             case ALG_ECSCHNORR_VALUE:
898                 return auth->signature.rsapss.hash;
899             case ALG_ECSCHNORR_VALUE:
900                 // SM2 and ECSCHNORR are optional
901         #endif // ALG_SM2
902     }
903     //all ECC signatures look the same
904     return auth->signature.ecdsa.hash;
# if ALG_ECDAA
    // Don't know how to verify an ECDAA signature
    case ALG_ECDAA_VALUE:
        break;
# endif
#endif // ALG_ECC

    case ALG_HMAC_VALUE:
        return auth->signature.hmac.hashAlg;
    default:
        break;
    }
    return ALG_NULL_VALUE;
}

10.2.6.6.10 CryptIsSplitSign()

This function is used to determine if the signing operation is a split signing operation that required a TPM2_Commit().

BOOL CryptIsSplitSign(TPM_ALG_ID scheme) // IN: the algorithm selector
{
    switch(scheme)
    {
    # if ALG_ECDAA
        case ALG_ECDAA_VALUE:
            return TRUE;
        break;
    # endif // ALG_ECDAA
    default:
        return FALSE;
        break;
    }
}

10.2.6.6.11 CryptIsAsymSignScheme()

This function indicates if a scheme algorithm is a sign algorithm.

BOOL CryptIsAsymSignScheme(TMĮ_ALG_PUBLIC publicType, // IN: Type of the object
                          TMĮ_ALG_ASM_SCHEME scheme  // IN: the scheme
)
{
    BOOL isSignScheme = TRUE;
    switch(publicType)
    {
    #if ALG_RSA
        case ALG_RSA_VALUE:
            switch(scheme)
            {
    # if !ALG_RSASSA || !ALG_RSAPSS
            # error "RSASSA and PSAPSS required if RSA used."
    # endif
            case ALG_RSASSA_VALUE:
                return ALG_NULL_VALUE;
            break;
    # endif // ALG_RSA
    default:
        return FALSE;
        break;
    }
    return TRUE;
}
961           case ALG_RSAPSS_VALUE:
962               break;
963           default:
964               isSignScheme = FALSE;
965               break;
966           }
967           break;
968       #endif // ALG_RSA
969   
970       #if ALG_ECC
971       // If ECC is implemented ECDSA is required
972       case ALG_ECC_VALUE:
973           switch(scheme)
974           {
975               // Support for ECDSA is required for ECC
976               case ALG_ECDSA_VALUE:
977                   #if ALG_ECDAA // ECDAA is optional
978                       case ALG_ECDAA_VALUE:
979                       #endif
980                   #if ALG_ECSCHNORR // Schnorr is also optional
981                       case ALG_ECSCHNORR_VALUE:
982                       #endif
983                   #if ALG_SM2 // SM2 is optional
984                       case ALG_SM2_VALUE:
985                       #endif
986               break;
987           default:
988               isSignScheme = FALSE;
989               break;
990           }
991           break;
992       #endif // ALG_ECC
993   
994       default:
995           isSignScheme = FALSE;
996           break;
997   
998   

10.2.6.6.12 CryptIsAsymDecryptScheme()

This function indicates if a scheme algorithm is a decrypt algorithm.

999
1000     BOOL CryptIsAsymDecryptScheme(
1001         TPMI_ALG_PUBLIC          publicType,   // IN: Type of the object
1002         TPMI_ALG_ASYM_SCHEME     scheme       // IN: the scheme
1003     )
1004     {
1005         BOOL isDecryptScheme = TRUE;
1006         switch(publicType)
1007         {
1008             #if ALG_RSA
1009             case ALG_RSA_VALUE:
1010                 switch(scheme)
1011                 {
1012                     case ALG_RSAES_VALUE:
1013                     case ALG_OAEP_VALUE:
1014                         break;
1015                     default:
1016                         isDecryptScheme = FALSE;
1017                         break;
1018                 }
1019             #endif
break;
#endif // ALG_RSA

#if ALG_ECC
  // If ECC is implemented ECDH is required
  case ALG_ECC_VALUE:
    switch (scheme)
    {
      #if !ALG_ECDH
        # error "ECDH is required for ECC"
      #endif
      #if ALG_SM2
        case ALG_SM2_VALUE:
      #endif
      #if ALG_ECMQV
        case ALG_ECMQV_VALUE:
      #endif
        break;
      default:
        isDecryptScheme = FALSE;
        break;
    }
    break;
#endif // ALG_ECC

default:
  isDecryptScheme = FALSE;
  break;
}

return isDecryptScheme;

10.2.6.6.13 CryptSelectSignScheme()

This function is used by the attestation and signing commands. It implements the rules for selecting the
signature scheme to use in signing. This function requires that the signing key either be TPM_RH_NULL
or be loaded.

If a default scheme is defined in object, the default scheme should be chosen, otherwise, the input
scheme should be chosen. In the case that both object and input scheme has a non-NULL scheme
algorithm, if the schemes are compatible, the input scheme will be chosen.

This function should not be called if 'signObject->publicArea.type' == ALG_SYMCIPHER.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>scheme selected</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>both scheme and key's default scheme are empty; or scheme is empty while key's default scheme requires explicit input scheme (split signing); or non-empty default key scheme differs from scheme</td>
</tr>
</tbody>
</table>

BOOL CryptSelectSignScheme(
  OBJECT *signObject,  // IN: signing key
  TPMT_SIG_SCHEME *scheme   // IN/OUT: signing scheme
)
{
  TPMT_SIG_SCHEME *objectScheme;
  TPMT_PUBLIC    *publicArea;
  BOOL           OK;

  // If the signHandle is TPM_RH_NULL, then the NULL scheme is used, regardless
  // of the setting of scheme
if (signObject == NULL)
{
    OK = TRUE;
    scheme->scheme = ALG_NULL_VALUE;
    scheme->details.any.hashAlg = ALG_NULL_VALUE;
}
else
{
    // assignment to save typing.
    publicArea = &signObject->publicArea;

    // A symmetric cipher can be used to encrypt and decrypt but it can't
    // be used for signing
    if (publicArea->type == ALG_SYMCIPHER_VALUE)
        return FALSE;

    // Point to the scheme object
    if (CryptIsAsymAlgorithm(publicArea->type))
        objectScheme =
            (TPMT_SIG_SCHEME *)&publicArea->parameters.asymDetail.scheme;
    else
        objectScheme =
            (TPMT_SIG_SCHEME *)&publicArea->parameters.keyedHashDetail.scheme;

    // If the object doesn't have a default scheme, then use the
    // input scheme.
    if (objectScheme->scheme == ALG_NULL_VALUE)
    {
        // Input and default can't both be NULL
        OK = (scheme->scheme != ALG_NULL_VALUE);
        // Assume that the scheme is compatible with the key. If not,
        // an error will be generated in the signing operation.
    }
    else if (scheme->scheme == ALG_NULL_VALUE)
    {
        // input scheme is NULL so use default
        // First, check to see if the default requires that the caller
        // provided scheme data
        OK = !CryptIsSplitSign(objectScheme->scheme);
        if (OK)
        {
            // The object has a scheme and the input is TPM_ALG_NULL so copy
            // the object scheme as the final scheme. It is better to use a
            // structure copy than a copy of the individual fields.
            *scheme = *objectScheme;
        }
    }
    else
    {
        // Both input and object have scheme selectors
        // If the scheme and the hash are not the same then...
        // NOTE: the reason that there is no copy here is that the input
        // might contain extra data for a split signing scheme and that
        // data is not in the object so, it has to be preserved.
        OK = (objectScheme->scheme == scheme->scheme)
            && (objectScheme->details.any.hashAlg
                == scheme->details.any.hashAlg);
    }
}
return OK;
10.2.6.6.14 CryptSign()

Sign a digest with asymmetric key or HMAC. This function is called by attestation commands and the generic TPM2_Sign() command. This function checks the key scheme and digest size. It does not check if the sign operation is allowed for restricted key. It should be checked before the function is called. The function will assert if the key is not a signing key.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SCHEME</td>
<td><code>signScheme</code> is not compatible with the signing key type</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td><code>digest</code> value is greater than the modulus of <code>signHandle</code> or size of</td>
</tr>
<tr>
<td></td>
<td><code>hashData</code> does not match hash algorithm <code>insignScheme</code> (for an RSA key);</td>
</tr>
<tr>
<td></td>
<td>invalid commit status or failed to generate <code>r</code> value (for an ECC key)</td>
</tr>
</tbody>
</table>

```c
TPM_RC
CryptSign(
    OBJECT              *signKey,       // IN: signing key
    TPM_SIG_SCHEME     *signScheme,     // IN: sign scheme.
    TPM2B_DIGEST        *digest,        // IN: The digest being signed
    TPM_SIGNATURE      *signature       // OUT: signature
) {
    TPM_RC               result = TPM_RC_SCHEME;
    // Initialize signature scheme
    signature->sigAlg = signScheme->scheme;
    // If the signature algorithm is TPM_ALG_NULL or the signing key is NULL,
    // then we are done
    if((signature->sigAlg == ALG_NULL_VALUE) || (signKey == NULL))
        return TPM_RC_SUCCESS;
    // Initialize signature hash
    // Note: need to do the check for TPM_ALG_NULL first because the null scheme
    // doesn't have a hashAlg member.
    signature->signature.any.hashAlg = signScheme->details.any.hashAlg;
    // perform sign operation based on different key type
    switch(signKey->publicArea.type) {
        #if ALG_RSA
            case ALG_RSA_VALUE:
                result = CryptRsaSign(signature, signKey, digest, NULL);
                break;
        #endif // ALG_RSA
        #if ALG_ECC
            case ALG_ECC_VALUE:
                // The reason that `signScheme` is passed to CryptEccSign but not to the
                // other signing methods is that the signing for ECC may be split and
                // need the 'r' value that is in the scheme but not in the signature.
                result = CryptEccSign(signature, signKey, digest,
                    (TPMT_ECC_SCHEME *)signScheme, NULL);
                break;
        #endif // ALG_ECC
        #if ALG_KEYEDHASH
            case ALG_KEYEDHASH_VALUE:
                result = CryptHmacSign(signature, signKey, digest);
                break;
        #endif // ALG_KEYEDHASH
        default:
            FAIL(FATAL_ERROR_INTERNAL);
            break;
    }
    return result;
}
```

10.2.6.6.14 CryptSign()
10.2.6.6.15 CryptValidateSignature()

This function is used to verify a signature. It is called by TPM2_VerifySignature() and TPM2_PolicySigned().

Since this operation only requires use of a public key, no consistency checks are necessary for the key to signature type because a caller can load any public key that they like with any scheme that they like. This routine simply makes sure that the signature is correct, whatever the type.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIGNATURE</td>
<td>the signature is not genuine</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>the scheme is not supported</td>
</tr>
<tr>
<td>TPM_RC_HANDLE</td>
<td>an HMAC key was selected but the private part of the key is not loaded</td>
</tr>
</tbody>
</table>

```c
TPM_RC CryptValidateSignature(
    TPMI_DH_OBJECT   keyHandle,       // IN: The handle of sign key
    TPM2B_DIGEST    *digest,         // IN: The digest being validated
    TPM_SIG                     *signature // IN: signature
)
{
    // NOTE: HandleToObject will either return a pointer to a loaded object or
    // will assert. It will never return a non-valid value. This makes it save
    // to initialize 'publicArea' with the return value from HandleToObject()
    // without checking it first.
    OBJECT              *signObject = HandleToObject(keyHandle);
    TPM_PUBLIC         *publicArea = &signObject->publicArea;
    TPM_RC               result = TPM_RC_SCHEME;

    // The input unmarshaling should prevent any input signature from being
    // a NULL signature, but just in case
    if(signature->sigAlg == ALG_NULL_VALUE)
        return TPM_RC_SIGNATURE;

    switch(publicArea->type)
    {
        #if ALG_RSA
            case ALG_RSA_VALUE:
                { // Call RSA code to verify signature
                    result = CryptRSAValidateSignature(signature, signObject, digest);
                break;
        }
        #endif // ALG_RSA

        #if ALG_ECC
            case ALG_ECC_VALUE:
                result = CryptEccValidateSignature(signature, signObject, digest);
            break;
        #endif // ALG_ECC

        case ALG_KEYEDHASH_VALUE:
            if(signObject->attributes.publicOnly)
                result = TPM_RC_HANDLE;
            else
                result = CryptHMACVerifySignature(signObject, digest, signature);
            break;
        default:
            break;
    }

    return result;
}
```

```
1220     return result;
1221 }

10.2.6.6.16 CryptGetTestResult

This function returns the results of a self-test function.

NOTE: The behavior in this function is NOT the correct behavior for a real TPM implementation. An artificial behavior is placed here due to the limitation of a software simulation environment. For the correct behavior, consult the part 3 specification for TPM2_GetTestResult().

```

```
1222     TPM_RC
1223     CryptGetTestResult(
1224         TPM2B_MAX_BUFFER    *outData       // OUT: test result data
1225     )
1226 {
1227     outData->t.size = 0;
1228     return TPM_RC_SUCCESS;
1229 }

10.2.6.6.17 CryptIsUniqueSizeValid()

This function validates that the unique values are consistent.

NOTE: This is not a comprehensive test of the public key.

```

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>sizes are consistent</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>sizes are not consistent</td>
</tr>
</tbody>
</table>

```

```
1230     BOOL
1231     CryptIsUniqueSizeValid(
1232         TPMT_PUBLIC     *publicArea       // IN: the public area to check
1233     )
1234 {
1235     BOOL            consistent = FALSE;
1236     UINT16          keySizeInBytes;
1237     switch(publicArea->type)
1238     {
1239         #if ALG_RSA
1240             case ALG_RSA_VALUE:
1241                 keySizeInBytes = BITS_TO_BYTES(
1242                     publicArea->parameters.rsaDetail.keyBits);
1243                 consistent = publicArea->unique.rsa.t.size == keySizeInBytes;
1244                 break;
1245         #endif // ALG_RSA
1246         #if ALG_ECC
1247             case ALG_ECC_VALUE:
1248                 { keySizeInBytes = BITS_TO_BYTES(CryptEccGetKeySizeForCurve(
1249                     publicArea->parameters.eccDetail.curveID));
1250                 consistent = keySizeInBytes > 0
1251                     && publicArea->unique.ecc.x.t.size <= keySizeInBytes
1252                     && publicArea->unique.ecc.y.t.size <= keySizeInBytes;
1253                 break;
1254         #endif // ALG_ECC
1255         default:
1256             // For SYMCIPHER and KEYEDHASH objects, the unique field is the size
```
1260       // of the nameAlg digest.
1261       consistent = publicArea->unique.sym.t.size
1262       == CryptHashGetDigestSize(publicArea->nameAlg);
1263       break;
1264   }
1265   return consistent;
1266 }

10.2.6.6.18 CryptIsSensitiveSizeValid()

This function is used by TPM2_LoadExternal() to validate that the sensitive area contains a sensitive value that is consistent with the values in the public area.

1267    BOOL
1268    CryptIsSensitiveSizeValid(
1269     TPM_PUBLIC             *publicArea,    // IN: the object’s public part
1270     TPM_SENSITIVE          *sensitiveArea   // IN: the object’s sensitive part
1271    )
1272  {
1273     BOOL                     consistent;
1274     UINT16                   keySizeInBytes;
1275
1276     switch (publicArea->type)
1277     {
1278         #if ALG_RSA
1279             case ALG_RSA_VALUE:
1280                 // sensitive prime value has to be half the size of the public modulus
1281                 keySizeInBytes = BITS_TO_BYTES(publicArea->parameters.rsaDetail.keyBits);
1282                 consistent = ((sensitiveArea->sensitive.rsa.t.size * 2) == keySizeInBytes);
1283                 break;
1284         #endif
1285         #if ALG_ECC
1286             case ALG_ECC_VALUE:
1287                 keySizeInBytes = BITS_TO_BYTES(CryptEccGetKeySizeForCurve(
1288                     publicArea->parameters.eccDetail.curveID));
1289                 consistent = (keySizeInBytes > 0)
1290                     && (sensitiveArea->sensitive.ecc.t.size == keySizeInBytes);
1291                 break;
1292         #endif
1293         case ALG_SYMCIPHER_VALUE:
1294             keySizeInBytes = BITS_TO_BYTES(publicArea->parameters.symDetail.sym.keyBits.sym);
1295             consistent = keySizeInBytes == sensitiveArea->sensitive.sym.t.size;
1296             break;
1297         case ALG_KEYEDHASH_VALUE:
1298             keySizeInBytes = CryptHashGetBlockSize(publicArea->nameAlg);
1299             // if the block size is 0, then the algorithm is TPM_ALG_NULL and the
1300             // size of the private part is limited to 128. If the algorithm block
1301             // size is over 128 bytes, then the size is limited to 128 bytes for
1302             // interoperability reasons.
1303             if((keySizeInBytes == 0) || (keySizeInBytes > 128))
1304                 keySizeInBytes = 128;
1305             consistent = sensitiveArea->sensitive.bits.t.size <= keySizeInBytes;
1306             break;
1307         default:
1308             consistent = TRUE;
1309             break;
1310     }
1311     return consistent;
1312 }

10.2.6.19  CryptValidateKeys()

This function is used to verify that the key material of an object is valid. For a publicOnly object, the key is verified for size and, if it is an ECC key, it is verified to be on the specified curve. For a key with a sensitive area, the binding between the public and private parts of the key are verified. If the nameAlg of the key is TPM_ALG_NULL, then the size of the sensitive area is verified but the public portion is not verified, unless the key is an RSA key. For an RSA key, the reason for loading the sensitive area is to use it. The only way to use a private RSA key is to compute the private exponent. To compute the private exponent, the public modulus is used.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC BINDING</td>
<td>the public and private parts are not cryptographically bound</td>
</tr>
<tr>
<td>TPM_RC HASH</td>
<td>cannot have a publicOnly key with nameAlg of TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPM_RC KEY</td>
<td>the public unique is not valid</td>
</tr>
<tr>
<td>TPM_RC KEY_SIZE</td>
<td>the private area key is not valid</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>the types of the sensitive and private parts do not match</td>
</tr>
</tbody>
</table>

```
TPM_RC
CryptValidateKeys(
    TPMT_PUBLIC *publicArea,
    TPMT_SENSITIVE *sensitive,
    TPM_RC blamePublic,
    TPM_RC blameSensitive
)
{
    TPM_RC result;
    UINT16 keySizeInBytes;
    UINT16 digestSize = CryptHashGetDigestSize(publicArea->nameAlg);
    TPMU_PUBLIC_PARMS *params = &publicArea->parameters;
    TPMU_PUBLIC_ID *unique = &publicArea->unique;
    if (sensitive != NULL)
    {
        // Make sure that the types of the public and sensitive are compatible
        if (publicArea->type != sensitive->sensitiveType)
            return TPM_RCS_TYPE + blameSensitive;
        // Make sure that the authValue is not bigger than allowed
        // If there is no name algorithm, then the size just needs to be less than
        // the maximum size of the buffer used for authorization. That size check
        // was made during unmarshaling of the sensitive area
        if ((sensitive->authValue.t.size) > digestSize && (digestSize > 0))
            return TPM_RCS_SIZE + blameSensitive;
    }
    switch (publicArea->type)
    {
        #if ALG_RSA
            case ALG_RSA_VALUE:
                keySizeInBytes = BITS_TO_BYTES(rsaDetail.keyBits);
                // Regardless of whether there is a sensitive area, the public modulus
                // needs to have the correct size. Otherwise, it can't be used for
                // any public key operation nor can it be used to compute the private
                // exponent.
                // NOTE: This implementation only supports key sizes that are multiples
                // of 1024 bits which means that the MSb of the 0th byte will always be
                // SET in either a prime or the public modulus.
                if ((unique->rsa.t.size != keySizeInBytes)
                    || (unique->rsa.t.buffer[0] < 0x80))
                    return TPM_RCS_KEY + blamePublic;
                if (params->rsaDetail.exponent != 0)
                    return TPM_RCS_KEY + blamePublic;
```

```
& params->rsaDetail.exponent < 7)
    return TPM_RCS_VALUE + blamePublic;
if(sensitive != NULL)
{
    // If there is a sensitive area, it has to be the correct size
    // including having the correct high order bit SET.
    if(((sensitive->敏感 rsa t.size * 2) != keySizeInBytes)
        | (sensitive->敏感 rsa t.buffer[0] < 0x80))
        return TPM_RCS_KEY_SIZE + blameSensitive;
}
break;
#endif
#if ALG_ECC
case ALG_ECC_VALUE:
{
    TPMI_ECC_CURVE curveId;
    curveId = params->eccDetail.curveID;
    keySizeInBytes = BITS_TO_BYTES(CryptEccGetKeySizeForCurve(curveId));
    if(sensitive == NULL)
    {
        // Validate the public key size
        if(unique->ecc.x.t.size != keySizeInBytes
            | unique->ecc.y.t.size != keySizeInBytes)
            return TPM_RCS_KEY + blamePublic;
        if(publicArea->nameAlg != ALG_NULL_VALUE)
        {
            if(!CryptEccIsPointOnCurve(curveId, &unique->ecc))
                return TPM_RCS_ECC_POINT + blamePublic;
        }
        else
        {
            // If the nameAlg is TPM_ALG_NULL, then only verify that the
            // private part of the key is OK.
            if(!CryptEccIsValidPrivateKey(&sensitive->sensitive.ecc,
                curveId))
                return TPM_RCS_KEY_SIZE;
            if(publicArea->nameAlg != ALG_NULL_VALUE)
            {
                // Full key load, verify that the public point belongs to the
                // private key.
                TPMI_ECC_POINT toCompare;
                result = CryptEccPointMultiply(&toCompare, curveId, NULL,
                    &sensitive->sensitive.ecc, NULL, NULL);
                if(result != TPM_RC_SUCCESS)
                    return TPM_RCS_BINDING;
                else
                {
                    // Make sure that the private key generated the public key.
                    // The input values and the values produced by the point
                    // multiply may not be the same size so adjust the computed
                    // value to match the size of the input value by adding or
                    // removing zeros.
                    AdjustNumberB(&toCompare.x.b, unique->ecc.x.t.size);
                    AdjustNumberB(&toCompare.y.b, unique->ecc.y.t.size);
                    if(!MemoryEqual2B(&unique->ecc.x.b, &toCompare.x.b)
                        | !MemoryEqual2B(&unique->ecc.y.b, &toCompare.y.b))
                        return TPM_RCS_BINDING;
                }
            }
        }
    }
else
{ /* default: */
}
// Checks for SYMCIPHER and KEYEDHASH are largely the same
// If public area has a nameAlg, then validate the public area size
// and if there is also a sensitive area, validate the binding

// For consistency, if the object is public-only just make sure that
// the unique field is consistent with the name algorithm
if(sensitive == NULL)
{
    if(unique->sym.t.size != digestSize)
        return TPM_RCS_KEY + blamePublic;
} else
{
    // Make sure that the key size in the sensitive area is consistent.
    if(publicArea->type == ALG_SYMCIPHER_VALUE)
    {
        result = CryptSymKeyValidate(&params->symDetail.sym,
            &sensitive->sensitive.sym);
        if(result != TPM_RC_SUCCESS)
            return result + blameSensitive;
    } else
    {
        // For a keyed hash object, the key has to be less than the
        // smaller of the block size of the hash used in the scheme or
        // 128 bytes. The worst case value is limited by the
        // unmarshaling code so the only thing left to be checked is
        // that it does not exceed the block size of the hash.
        // by the hash algorithm of the scheme.
        TPMKEYEDHASHSCHEME *scheme;
        UINT16 maxSize;
        scheme = &params->keyedHashDetail.scheme;
        if(scheme->scheme == ALG_XOR_VALUE)
        {
            maxSize = CryptHashGetBlockSize(scheme->details.xor.hashAlg);
        } else if(scheme->scheme == ALG_HMAC_VALUE)
        {
            maxSize = CryptHashGetBlockSize(scheme->details.hmac.hashAlg);
        } else if(scheme->scheme == ALG_NULL_VALUE)
        {
            // Not signing or xor so must be a data block
            maxSize = 128;
        } else
        {
            return TPM_RCS_SCHEME + blamePublic;
        }
        if(sensitive->sensitive.bits.t.size > maxSize)
            return TPM_RCS_KEY_SIZE + blameSensitive;
    }
    // If there is a nameAlg, check the binding
    if(publicArea->nameAlg != ALG_NULL_VALUE)
    {
        TPM2BDIGEST compare;
        if(sensitive->seedValue.t.size != digestSize)
            return TPM_RCS_KEY_SIZE + blameSensitive;
        CryptComputeSymmetricUnique(publicArea, sensitive, &compare);
        if(!MemoryEqual2B(unique->sym.b, &compare.b))
            return TPM_RC_BINDING;
    }
}
break;
// For a parent, need to check that the seedValue is the correct size for
// protections. It should be at least half the size of the nameAlg
if(IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, restricted)  
&& IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, decrypt)  
&& sensitive != NULL  
&& publicArea->nameAlg != ALG_NULL_VALUE)  
{
  if((sensitive->seedValue.t.size < (digestSize / 2))  
    || (sensitive->seedValue.t.size > digestSize))
    return TPM_RCS_SIZE + blameSensitive;
  return TPM_RC_SUCCESS;
}

10.2.6.6.20 CryptAlgSetImplemented()

This function initializes the bit vector with one bit for each implemented algorithm. This function is called from _TPM_Init(). The vector of implemented algorithms should be generated by the part 2 parser so that the g_implementedAlgorithms vector can be a constant. That's not how it is now.

```c
void CryptAlgsSetImplemented(
  void
)
{
  AlgorithmGetImplementedVector(&g_implementedAlgorithms);
}
```

10.2.6.6.21 CryptSelectMac()

This function is used to set the MAC scheme based on the key parameters and the input scheme.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SCHEME</td>
<td>the scheme is not a valid mac scheme</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>the input key is not a type that supports a mac</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>the input scheme and the key scheme are not compatible</td>
</tr>
</tbody>
</table>

```c
TPM_RC
CryptSelectMac(  
  TPMT_PUBLIC             *publicArea,  
  TPMI_ALG_MAC_SCHEME     *inMac  
)
{
  TPM_ALG_ID              macAlg = ALG_NULL_VALUE;
  switch(publicArea->type)
  {
    case ALG_KEYEDHASH_VALUE:
      {
        // Local value to keep lines from getting too long
        TPMT_KEYEDHASH_SCHEME   *scheme;
        scheme = &publicArea->parameters.keyedHashDetail.scheme;
        // Expect that the scheme is either HMAC or NULL
        if(scheme->scheme != ALG_NULL_VALUE)  
          macAlg = scheme->details.hmac.hashAlg;
        break;
      }
    case ALG_SYMCIPHER_VALUE:
      {
        // Expect that the scheme is either valid symmetric cipher or NULL
        TPMT_SYM_DEF_OBJECT     *scheme;
        scheme = &publicArea->parameters.symDetail.sym;
        if(scheme->algorithm != ALG_NULL_VALUE)
```
macAlg = scheme->mode.sym;
break;
}
default:
    return TPM_RCS_TYPE;
}

// If the input value is not TPM_ALG_NULL ...
if(*inMac != ALG_NULL_VALUE)
{
    // ... then either the scheme in the key must be TPM_ALG_NULL or the input
    // value must match
    if((macAlg != ALG_NULL_VALUE) && (*inMac != macAlg))
    return TPM_RCS_VALUE;
}
else
{
    // Since the input value is TPM_ALG_NULL, then the key value can't be
    // ALG NULL
    if(macAlg == ALG_NULL_VALUE)
    return TPM_RCS_VALUE;
    *inMac = macAlg;
}

if(!CryptMacIsValidForKey(publicArea->type, *inMac, FALSE))
return TPM_RCS_SCHEME;
return TPM_RC_SUCCESS;

10.2.6.6.22 CryptMacIsValidForKey()

Check to see if the key type is compatible with the mac type

BOOL CryptMacIsValidForKey(
    TPM_ALG_ID keyType,
    TPM_ALG_ID macAlg,
    BOOL flag
)
{
    switch(keyType)
    {
    case ALG_KEYEDHASH_VALUE:
        return CryptHashIsValidAlg(macAlg, flag);
        break;
    case ALG_SYMCIPHER_VALUE:
        return CryptSmacIsValidAlg(macAlg, flag);
        break;
    default:
        break;
    }
    return FALSE;
}

10.2.6.6.23 CryptSmacIsValidAlg()

This function is used to test if an algorithm is a supported SMAC algorithm. It needs to be updated as new algorithms are added.

BOOL CryptSmacIsValidAlg(
    TPM_ALG_ID alg,
    BOOL flag // IN: Indicates if TPM_ALG_NULL is valid
)
{


switch (alg)
{
    #if ALG_CMAC
    case ALG_CMAC_VALUE:
        return TRUE;
        break;
    #endif
    case ALG_NULL_VALUE:
        return FLAG;
        break;
    default:
        return FALSE;
}

10.2.6.6.24 CryptSymModeIsValid()

Function checks to see if an algorithm ID is a valid, symmetric block cipher mode for the TPM. If flag is SET, then TPM_ALG_NULL is a valid mode. Not include the modes used for SMAC

BOOL CryptSymModeIsValid(
    TPM_ALG_ID mode,
    BOOL flag
)
{
    switch (mode)
    {
        #if ALG_CTR
        case ALG_CTR_VALUE:
        #endif // ALG_CTR
        #if ALG_OFB
        case ALG_OFB_VALUE:
        #endif // ALG_OFB
        #if ALG_CBC
        case ALG_CBC_VALUE:
        #endif // ALG_CBC
        #if ALG_CFB
        case ALG_CFB_VALUE:
        #endif // ALG_CFB
        #if ALG_ECB
        case ALG_ECB_VALUE:
        #endif // ALG_ECB
        return TRUE;
        case ALG_NULL_VALUE:
            return flag;
            break;
        default:
            break;
    }
    return FALSE;
10.2.7 CryptSelfTest.c

10.2.7.1 Introduction

The functions in this file are designed to support self-test of cryptographic functions in the TPM. The TPM allows the user to decide whether to run self-test on a demand basis or to run all the self-tests before proceeding.

The self-tests are controlled by a set of bit vectors. The g_untestedDecryptionAlgorithms vector has a bit for each decryption algorithm that needs to be tested and g_untestedEncryptionAlgorithms has a bit for each encryption algorithm that needs to be tested. Before an algorithm is used, the appropriate vector is checked (indexed using the algorithm ID). If the bit is 1, then the test function should be called.

For more information, see TpmSelfTests().txt

```c
#include "Tpm.h"
```

10.2.7.2 Functions

10.2.7.2.1 RunSelfTest()

Local function to run self-test

```c
static TPM_RC
CryptRunSelfTests(
    ALGORITHM_VECTOR    *toTest  // IN: the vector of the algorithms to test

) {
    TPM_ALG_ID alg;

    // For each of the algorithms that are in the toTestVecor, need to run a
    // test
    for(alg = TPM_ALG_FIRST; alg <= TPM_ALG_LAST; alg++)
    {
        if(TEST_BIT(alg, *toTest))
            {
                TPM_RC result = CryptTestAlgorithm(alg, toTest);
                if(result != TPM_RC_SUCCESS)
                    return result;
            }
    }
    return TPM_RC_SUCCESS;
}
```

10.2.7.2.2 CryptSelfTest()

This function is called to start/complete a full self-test. If fullTest is NO, then only the untested algorithms will be run. If fullTest is YES, then g_untestedDecryptionAlgorithms is reinitialized and then all tests are run. This implementation of the reference design does not support processing outside the framework of a TPM command. As a consequence, this command does not complete until all tests are done. Since this can take a long time, the TPM will check after each test to see if the command is canceled. If so, then the TPM will returned TPM_RC_CANCELLED. To continue with the self-tests, call TPM2_SelfTest(fullTest == No) and the TPM will complete the testing.
### Error Returns

<table>
<thead>
<tr>
<th>Meaning</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>if the command is canceled</td>
<td>TPM_RC_CANCELED</td>
</tr>
</tbody>
</table>

```c
22 LIB_EXPORT
23 TPM_RC
24 CryptSelfTest(
25     TPMI_YES_NO fullTest, // IN: if full test is required
26 )
27 {
28     #if SIMULATION
29         if (g_forceFailureMode)
30             FAIL(FATAL_ERROR_FORCED);
31     #endif
32
33     // If the caller requested a full test, then reset the to test vector so that
34     // all the tests will be run
35     if (fullTest == YES)
36         {  // Transcribe the toTest list into the toTestVector
37             MemoryCopy(g_toTest,
38                 g_implementedAlgorithms,
39                 sizeof(g_toTest));
40             return CryptRunSelfTests(&g_toTest);
41         }
42 }
```

#### 10.2.7.2.3 CryptIncrementalSelfTest()

This function is used to perform an incremental self-test. This implementation will perform the `toTest` values before returning. That is, it assumes that the TPM cannot perform background tasks between commands.

This command may be canceled. If it is, then there is no return result. However, this command can be run again and the incremental progress will not be lost.

```c
43 TPM_RC
44 CryptIncrementalSelfTest(
45     TPMI_ALG *toTest,     // IN: list of algorithms to be tested
46     TPMI_ALG *tToTest,    // OUT: list of algorithms needing test
47 )
48 {
49     ALGORITHM_VECTOR toTestVector = {0};
50     TPM_ALG_ID alg;
51     UINT32 i;
52
53     pAssert(toTest != NULL && tToTest != NULL);
54     if(toTest->count > 0)
55         {
56             // Transcribe the toTest list into the toTestVector
57             for (i = 0; i < toTest->count; i++)
58                 {
59                     alg = toTest->algorithms[i];
60                     // make sure that the algorithm value is not out of range
61                     if ((alg > TPM_ALG_LAST) || !TEST_BIT(alg, g_implementedAlgorithms))
62                         continue;
63                     
64                     // Check that the toTest value is valid
65                     if (alg == TPM_ALG_LAST)
66                         toTestVector[i] = TPMI_YES;
67                     else
68                         toTestVector[i] = TPMI_NO;
69                 }
70             
71             TPM_RC_RC toTestError = TPM_RC_UNRECOGNIZED_ERROR;
72             for (i = 0; i < toTest->count; i++)
73                 
74                 
75             return toTestError;
76         }
77     return TPM_RC_SUCCESS;
78 }
```
return TPM_RC_VALUE;
    SET_BIT(alg, toTestVector);
}

// Run the test
if(CryptRunSelfTests(&toTestVector) == TPM_RC_CANCELED)
    return TPM_RC_CANCELED;
}

// Fill in the toDoList with the algorithms that are still untested
toDoList->count = 0;
for(alg = TPM_ALG_FIRST;
toDoList->count < MAX_ALG_LIST_SIZE && alg <= TPM_ALG_LAST;
alg++)
{
    if(TRAVERSE_BIT(alg, g_toTest))
        toDoList->algorithms[toDoList->count++] = alg;
}
return TPM_RC_SUCCESS;

10.2.7.2.4 CryptInitializeToTest()

This function will initialize the data structures for testing all the algorithms. This should not be called unless CryptAlgsSetImplemented() has been called

void CryptInitializeToTest()
{
    // Indicate that nothing has been tested
    memset(&g_cryptoSelfTestState, 0, sizeof(g_cryptoSelfTestState));

    // Copy the implemented algorithm vector
    MemoryCopy(g_toTest, g_implementedAlgorithms, sizeof(g_toTest));

    // Setting the algorithm to null causes the test function to just clear
    // out any algorithms for which there is no test.
    CryptTestAlgorithm(TPM_ALG_ERROR, &g_toTest);
    return;
}

10.2.7.2.5 CryptTestAlgorithm()

Only point of contact with the actual self tests. If a self-test fails, there is no return and the TPM goes into failure mode. The call to TestAlgorithm() uses an algorithm selector and a bit vector. When the test is run, the corresponding bit in toTest and in g_toTest is CLEAR. If toTest is NULL, then only the bit in g_toTest is CLEAR. There is a special case for the call to TestAlgorithm(). When alg is ALG_ERROR, TestAlgorithm() will CLEAR any bit in toTest for which it has no test. This allows the knowledge about which algorithms have test to be accessed through the interface that provides the test.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CANCELED</td>
<td>test was canceled</td>
</tr>
</tbody>
</table>
104 }  
105 {  
106 TPM_RC result;  
107 #if SELF_TEST  
108 result = TestAlgorithm(alg, toTest);  
109 #else  
110 // If this is an attempt to determine the algorithms for which there is a  
111 // self test, pretend that all of them do. We do that by not clearing any  
112 // of the algorithm bits. When/if this function is called to run tests, it  
113 // will over report. This can be changed so that any call to check on which  
114 // algorithms have tests, 'toTest' can be cleared.  
115 if(alg != TPM_ALG_ERROR)  
116 {  
117 #if SELF_TEST  
118 if(toTest != NULL)  
119 CLEAR_BIT(alg, *toTest);  
120 }  
121 // If this is an attempt to determine the algorithms for which there is a  
122 // self test, pretend that all of them do. We do that by not clearing any  
123 // of the algorithm bits. When/if this function is called to run tests, it  
124 // will over report. This can be changed so that any call to check on which  
125 // algorithms have tests, 'toTest' can be cleared.  
126 result = TPM_RC_SUCCESS;  
127 #endif  
128 return result;  
129 }
10.2.8  CryptEccData.c

This file contains the ECC curve data. The format of the data depends on the setting of USE_BN_ECC_DATA. If it is defined, then the TPM's BigNum() format is used. Otherwise, it is kept in TPM2B format. The purpose of having the data in BigNum() format is so that it does not have to be reformatted before being used by the crypto library.

```
#include "Tpm.h"

#define BN_MIN_ALLOC(bytes) ((BYTES_TO_CRYPT_WORDS(bytes) == 0) ? 1 : BYTES_TO_CRYPT_WORDS(bytes))

#define ECC_CONST(NAME, bytes, initializer) const struct {
    crypt_uword_t allocate, size, d[BN_MIN_ALLOC(bytes)];
} NAME = {bytes, {initializer}}
```

define how to transform a curve parameter address into an entry into an ECC_CURVE_DATA structure.

```
#define ECC_ENTRY(val, x) (bigNum)&val##_##x
```

Have to special case ECC ZERO

```
TPM2B_BYTE_VALUE(1);
TPM2B_1_BYTE_VALUE ECC ZERO = {1, {0}};
```

```
#define ECC_ENTRY(val, x) &val##_##x##.b
```

```
#if !USE_BN_ECC_DATA
TPM2B_BYTE_VALUE(24);
#define TO_ECC_192(a, b, c) a, b, c
TPM2B_BYTE_VALUE(28);
#define TO_ECC_224(a, b, c, d) a, b, c, d
TPM2B_BYTE_VALUE(32);
#define TO_ECC_256(a, b, c, d) a, b, c, d
TPM2B_BYTE_VALUE(48);
#define TO_ECC_384(a, b, c, d, e, f) a, b, c, d, e, f
TPM2B_BYTE_VALUE(66);
#define TO_ECC_528(a, b, c, d, e, f, g, h, i) a, b, c, d, e, f, g, h, i
TPM2B_BYTE_VALUE(80);
#endif // !USE_BN_ECC_DATA
#if ECC_NIST_P192
ECC_CONST(NIST_P192_p, 24, TO_ECC_192(TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFE),
TO_ECC_64(0xFF, 0xFF, 0xFF, 0x00, 0x00, 0x00, 0x00, 0x01)));
ECC_CONST(NIST_P192_a, 24, TO_ECC_192(TO_ECC_64(0x18, 0x0B, 0x05, 0x85),
TO_ECC_64(0x0C, 0x04, 0xB3, 0xAB, 0xF5, 0x41, 0x32, 0x56),
TO_ECC_64(0x50, 0x44, 0xB0, 0xB7, 0xD7, 0xBF, 0xD8, 0xBA)),
ECC_CONST(NIST_P192_b, 24, TO_ECC_192(TO_ECC_64(0x18, 0x0B, 0x05, 0x85),
TO_ECC_64(0x0C, 0x04, 0xB3, 0xAB, 0xF5, 0x41, 0x32, 0x56),
TO_ECC_64(0x50, 0x44, 0xB0, 0xB7, 0xD7, 0xBF, 0xD8, 0xBA)),
ECC_CONST(NIST_P192_gX, 24, TO_ECC_192(TO_ECC_64(0x18, 0x0B, 0x05, 0x85),
TO_ECC_64(0x0C, 0x04, 0xB3, 0xAB, 0xF5, 0x41, 0x32, 0x56),
TO_ECC_64(0x50, 0x44, 0xB0, 0xB7, 0xD7, 0xBF, 0xD8, 0xBA)),
ECC_CONST(NIST_P192_gY, 24, TO_ECC_192(TO_ECC_64(0x07, 0x19, 0x2B, 0x95, 0xFF, 0xC8, 0xDA, 0x78),
TO_ECC_64(0x63, 0x10, 0x11, 0xED, 0x6B, 0x24, 0xCD, 0xD5),
TO_ECC_64(0x73, 0xF9, 0x77, 0xA1, 0x1E, 0x79, 0x48, 0x11)),
ECC_CONST(NIST_P192_n, 24, TO_ECC_192(TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
TO_ECC_64(0xFF, 0xFF, 0xFF, 0x99, 0xDE, 0xF8, 0x36),
TO_ECC_64(0x14, 0x6B, 0xC9, 0xB1, 0xB4, 0xD2, 0x28, 0x31)));
#define NIST_P192_h ECC_ONE
#define NIST_P192_gZ ECC_ONE
CURVE_SPEC(NIST_P192);
#endif // ECC_NIST_P192
#if ECC_NIST_P224
ECC_CONST(NIST_P224_p, 28, TO_ECC_224(TO_ECC_32(0xFF, 0xFF, 0xFF, 0xFF),
TO_ECC_64(0x07, 0x19, 0x2B, 0x95),
TO_ECC_64(0x63, 0x10, 0x11),
TO_ECC_64(0x73, 0xF9, 0x77, 0xA1),
TO_ECC_64(0x07, 0x19, 0x2B, 0x95),
TO_ECC_64(0x63, 0x10, 0x11),
TO_ECC_64(0x07, 0x19, 0x2B, 0x95),
TO_ECC_64(0x63, 0x10, 0x11)),
ECC_CONST(NIST_P224_a, 28, TO_ECC_224(TO_ECC_32(0xFF, 0xFF, 0xFF, 0xFF),
TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF),
TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF),
TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF),
TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF),
TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF)),
ECC_CONST(NIST_P224_b, 28, TO_ECC_224(TO_ECC_32(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00)),
ECC_CONST(NIST_P224_gX, 28, TO_ECC_224(TO_ECC_32(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00)),
ECC_CONST(NIST_P224_gY, 28, TO_ECC_224(TO_ECC_32(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00),
TO_ECC_64(0x00, 0x00, 0x00, 0x00)),
#define NIST_P224_h ECC_ONE
#define NIST_P224_gZ ECC_ONE
CURVE_SPEC(NIST_P224);
#endif // ECC_NIST_P224
ECC_CONST(NIST_P224_gX, 28, TO_ECC_224(
  TO_ECC_32(0xB7, 0x0E, 0xC0, 0xBD),
  TO_ECC_64(0x6B, 0xB4, 0xBF, 0x7F, 0x32, 0x13, 0x90, 0xB9),
  TO_ECC_64(0x4A, 0x03, 0xC1, 0xD3, 0x56, 0xC2, 0x11, 0x22),
  TO_ECC_64(0x34, 0x32, 0xB0, 0xD6, 0x11, 0x5C, 0x1D, 0x21)));

ECC_CONST(NIST_P224_gY, 28, TO_ECC_224(
  TO_ECC_32(0xB7, 0x37,
    0x63, 0x88),
  TO_ECC_64(0xB5, 0xF7, 0x23, 0xFB, 0x4C, 0x22, 0xDF, 0xE6),
  TO_ECC_64(0xCD, 0x43, 0x75, 0xA0, 0x5A, 0x07, 0x47, 0x64),
  TO_ECC_64(0x44, 0xD5, 0x81, 0x99, 0x85, 0x00, 0x7E, 0x34)));

ECC_CONST(NIST_P224_n, 28, TO_ECC_224(
  TO_ECC_32(0xFF, 0xFF, 0xFF, 0xFF),
  TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
  TO_ECC_64(0xFF, 0xFF, 0x16, 0xA2, 0xE0, 0xB8, 0xF0, 0x3E),
  TO_ECC_64(0x13, 0xDD, 0x29, 0x45, 0x5C, 0x5C, 0x2A, 0x3D)));

#define NIST_P224_h ECC_ONE
#define NIST_P224_gZ ECC_ONE
CURVE_SPEC(NIST_P224);
#endif // ECC_NIST_P224

#define NIST_P256_h ECC_ONE
#define NIST_P256_gZ ECC_ONE
CURVE_SPEC(NIST_P256);
#endif // ECC_NIST_P256

#define NIST_P384_h ECC_ONE
#define NIST_P384_gZ ECC_ONE
CURVE_SPEC(NIST_P384);
#endif // ECC_NIST_P384

Part 4: Supporting Routines

Trusted Platform Module Library

Level 00 Revision 01.50

September 18, 2018
ECC_CONST(NIST_P521_gX, 66, TO_ECC_528(
    TO_ECC_16(0x00, 0xC6),
    TO_ECC_64(0x85, 0x8E, 0x06, 0xB7, 0x04, 0x04, 0xE9, 0xCD),
    TO_ECC_64(0x9E, 0x3E, 0xCB, 0x66, 0x23, 0x95, 0xB4, 0x42),
    TO_ECC_64(0x9C, 0x81, 0x39, 0x05, 0x3F, 0xB5, 0x21),
    TO_ECC_64(0xF8, 0x28, 0xAF, 0x60, 0x6B, 0x4D, 0x3D, 0xBA),
    TO_ECC_64(0xA1, 0x4B, 0x5E, 0x77, 0xEF, 0xE7, 0x59, 0x28),
    TO_ECC_64(0xFE, 0x1D, 0xC1, 0x27, 0xA2, 0xFF, 0xA8, 0xDE),
    TO_ECC_64(0x33, 0x48, 0xB3, 0xC1, 0x85, 0x6A, 0x42, 0x9B),
    TO_ECC_64(0xF9, 0x7E, 0x7E, 0x31, 0xC2, 0xE5, 0xBD, 0x66)
));
ECC_CONST(NIST_P521_gY, 66, TO_ECC_528(
    TO_ECC_16(0x01, 0x18),
    TO_ECC_64(0x01, 0xFF),
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
    TO_ECC_64(0x01, 0xFF),
    TO_ECC_64(0x01, 0xFF),
    TO_ECC_64(0x01, 0xFF)
));
#define NIST_P521_h         ECC_ONE
#define NIST_P521_gZ        ECC_ONE
CURVE_SPEC(NIST_P521);
#endif // ECC_NIST_P521

#endif // ECC_BN_P256

#define BN_P256_a           ECC_ZERO
#define BN_P256_gX          ECC_ONE
#define BN_P256_gY          ECC_ONE
#define BN_P256_n           ECC_ONE
CURVE_SPEC(BN_P256);
#endif // ECC_BN_P256

#define BN_P638_a           ECC_ZERO
#define BN_P638_gX          ECC_ONE
#define BN_P638_gY          ECC_ONE
#define BN_P638_n           ECC_ONE
CURVE_SPEC(BN_P638);
#endif // ECC_BN_P638
ECC_CONST(BN_P638_b, 2, TO_ECC_16(0x01,0x01));
ECC_CONST(BN_P638_gX, 80, TO_ECC_640(
  TO_ECC_64(0x23, 0xFF, 0xFF, 0xFD, 0xC0, 0x00, 0x00, 0x0D),
  TO_ECC_64(0x7F, 0xFF, 0xFF, 0xB8, 0x00, 0x00, 0x01, 0xD3),
  TO_ECC_64(0xFF, 0xFF, 0xF9, 0x42, 0xD0, 0x00, 0x16, 0x5E),
  TO_ECC_64(0x3F, 0xFF, 0x94, 0x87, 0x00, 0x00, 0xD5, 0x2F),
  TO_ECC_64(0xFF, 0xFD, 0xE0, 0x00, 0x08, 0xDE, 0x55),
  TO_ECC_64(0xC0, 0x00, 0x86, 0x52, 0x00, 0x21, 0xE5, 0x5B),
  TO_ECC_64(0xFF, 0xFF, 0xF5, 0x1F, 0xFF, 0xF4, 0xEB, 0x80),
  TO_ECC_64(0x00, 0x00, 0x00, 0x4C, 0x80, 0x01, 0x5A, 0xCD),
  TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xEC, 0xE0),
  TO_ECC_64(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x66)));
ECC_CONST(BN_P638_gY, 1, TO_ECC_8(0x10));
ECC_CONST(BN_P638_n, 80, TO_ECC_640(
  #define BN_P638_h ECC_ONE
  #define BN_P638_gZ ECC_ONE
  CURVE_SPEC(BN_P638);
  #endif // ECC_BN_P638
  #if ECC_SM2_P256
  ECC_CONST(SM2_P256_p, 32, TO_ECC_256(
    TO_ECC_64(0x23, 0xFF, 0xFF, 0xFD, 0xC0, 0x00, 0x00, 0x0D),
    TO_ECC_64(0x7F, 0xFF, 0xFF, 0xB8, 0x00, 0x00, 0x01, 0xD3),
    TO_ECC_64(0xFF, 0xFF, 0xF9, 0x42, 0xD0, 0x00, 0x16, 0x5E),
    TO_ECC_64(0x3F, 0xFF, 0x94, 0x87, 0x00, 0x00, 0xD5, 0x2F),
    TO_ECC_64(0xFF, 0xFD, 0xE0, 0x00, 0x08, 0xDE, 0x55),
    TO_ECC_64(0xC0, 0x00, 0x86, 0x52, 0x00, 0x21, 0xE5, 0x5B),
    TO_ECC_64(0xFF, 0xFF, 0xF5, 0x1F, 0xFF, 0xF4, 0xEB, 0x80),
    TO_ECC_64(0x00, 0x00, 0x00, 0x4C, 0x80, 0x01, 0x5A, 0xCD),
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xEC, 0xE0),
    TO_ECC_64(0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x66)));
  ECC_CONST(SM2_P256_a, 32, TO_ECC_256(
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF),
    TO_ECC_64(0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF)),
  ECC_CONST(SM2_P256_b, 32, TO_ECC_256(
    TO_ECC_64(0x28, 0x89, 0xEA, 0x8E, 0x9D, 0x9F, 0x5E, 0x34),
    TO_ECC_64(0x4D, 0x5A, 0x9E, 0x4B, 0xCF, 0x65, 0x09, 0xA7),
    TO_ECC_64(0x97, 0x89, 0xFF, 0x15, 0x1F, 0x0F, 0x8F, 0x92),
    TO_ECC_64(0x71, 0x5A, 0x45, 0x89, 0x33, 0x4C, 0x74, 0xC7)),
  ECC_CONST(SM2_P256_gX, 32, TO_ECC_256(
    TO_ECC_64(0x32, 0xC4, 0xA4, 0x2C, 0x1F, 0x19, 0x81, 0x19),
    TO_ECC_64(0x5F, 0x99, 0x04, 0x46, 0x6A, 0x39, 0xC9, 0x94),
    TO_ECC_64(0x8F, 0xE3, 0x0B, 0xBF, 0xF2, 0x66, 0x0B, 0xE1),
    TO_ECC_64(0x71, 0x5A, 0x45, 0x89, 0x33, 0x4C, 0x74, 0xC7))),
  ECC_CONST(SM2_P256_gY, 32, TO_ECC_256(
    TO_ECC_64(0x28, 0x89, 0xEA, 0x8E, 0x9D, 0x9F, 0x5E, 0x34),
    TO_ECC_64(0x4D, 0x5A, 0x9E, 0x4B, 0xCF, 0x65, 0x09, 0xA7),
    TO_ECC_64(0x97, 0x89, 0xFF, 0x15, 0x1F, 0x0F, 0x8F, 0x92),
    TO_ECC_64(0x71, 0x5A, 0x45, 0x89, 0x33, 0x4C, 0x74, 0xC7))),
  #define SM2_P256_h ECC_ONE
  #define SM2_P256_gZ ECC_ONE
  CURVE_SPEC(SM2_P256);
  #endif // ECC_SM2_P256
  const ECC_CURVE eccCurves[] = {
    #if ECC_NIST_P192
    ECC_BN_P192

Part 4: Supporting Routines

Level 00 Revision 01.50

TCG Public Review

September 18, 2018
380     {TPM_ECC_NIST_P192,
381         192,
382         {ALG_KDF1_SP800_56A_VALUE, {{ALG_SHA256_VALUE}}},
383         {ALG_NULL_VALUE, {{ALG_NULL_VALUE}}},
384         &NIST_P192
385         CURVE_NAME("NIST_P192")}  
386     # undef comma
387     # define comma ,
388     #end if // ECC_NIST_P192
389     #if ECC_NIST_P224
390     comma
391     {TPM_ECC_NIST_P224,
392         224,
393         {ALG_KDF1_SP800_56A_VALUE, {{ALG_SHA256_VALUE}}},
394         {ALG_NULL_VALUE, {{ALG_NULL_VALUE}}},
395         &NIST_P224
396         CURVE_NAME("NIST_P224")}
397     # undef comma
398     # define comma ,
399     #end if // ECC_NIST_P224
400     #if ECC_NIST_P256
401     comma
402     {TPM_ECC_NIST_P256,
403         256,
404         {ALG_KDF1_SP800_56A_VALUE, {{ALG_SHA256_VALUE}}},
405         {ALG_NULL_VALUE, {{ALG_NULL_VALUE}}},
406         &NIST_P256
407         CURVE_NAME("NIST_P256")}
408     # undef comma
409     # define comma ,
410     #end if // ECC_NIST_P256
411     #if ECC_NIST_P384
412     comma
413     {TPM_ECC_NIST_P384,
414         384,
415         {ALG_KDF1_SP800_56A_VALUE, {{ALG_SHA384_VALUE}}},
416         {ALG_NULL_VALUE, {{ALG_NULL_VALUE}}},
417         &NIST_P384
418         CURVE_NAME("NIST_P384")}  
419     # undef comma
420     # define comma ,
421     #end if // ECC_NIST_P384
422     #if ECC_NIST_P521
423     comma
424     {TPM_ECC_NIST_P521,
425         521,
426         {ALG_KDF1_SP800_56A_VALUE, {{ALG_SHA512_VALUE}}},
427         {ALG_NULL_VALUE, {{ALG_NULL_VALUE}}},
428         &NIST_P521
429         CURVE_NAME("NIST_P521")}
430     # undef comma
431     # define comma ,
432     #end if // ECC_NIST_P521
433     #if ECC_BN_P256
434     comma
435     {TPM_ECC_BN_P256,
436         256,
437         {ALG_NULL_VALUE, {{ALG_NULL_VALUE}}},
438         {ALG_NULL_VALUE, {{ALG_NULL_VALUE}}},
439         &BN_P256
440         CURVE_NAME("BN_P256")}
441     # undef comma
442     # define comma ,
443     #end if // ECC_BN_P256
444     #if ECC_BN_P638
# undef comma
#endif // ECC_SM2_P256
}
#endif
#endif // TPM_ALG_ECC
10.2.9  CryptDes.c

10.2.9.1  Introduction

This file contains the extra functions required for TDES.

10.2.9.2  Includes, Defines, and Typedefs

```c
#include "Tpm.h"
#if ALG_TDES
#define DES_NUM_WEAK 64
const UINT64 DesWeakKeys[DES_NUM_WEAK] = {
    0x0101010101010101, 0xFEFEFEFEFEFEFEFE, 0xE0E0E0E0F1F1F1F1, 0x1F1F1F0E0E0E0E0,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x01FE1FFE1FFE0E0E, 0x01FE1FFE1FFE0E0E, 0xE0FE0E0E0F1F1F1F1, 0x1F0E0E0E0F1F1F1F1,
    0x01011F01010E010E, 0x1F01010E010E0101, 0xE001F1F1F1010E01, 0x01010E010E0101F1,
    0x1FFE0E0E0E1F1F1F1, 0x1F010E010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x01011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x011F011F010E010E, 0x011F011F010E010E, 0x1F011F010E010E01, 0x1F011F010E010E01,
    0x011F011F010E010E, 0x011F011F010E010E, 0x1F011F010E010E01, 0x1F011F010E010E01,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x011F011F010E010E, 0x011F011F010E010E, 0x1F011F010E010E01, 0x1F011F010E010E01,
    0x011F011F010E010E, 0x011F011F010E010E, 0x1F011F010E010E01, 0x1F011F010E010E01,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x011F011F010E010E, 0x011F011F010E010E, 0x1F011F010E010E01, 0x1F011F010E010E01,
    0x011F011F010E010E, 0x011F011F010E010E, 0x1F011F010E010E01, 0x1F011F010E010E01,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x011F011F010E010E, 0x011F011F010E010E, 0x1F011F010E010E01, 0x1F011F010E010E01,
    0x011F011F010E010E, 0x011F011F010E010E, 0x1F011F010E010E01, 0x1F011F010E010E01,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,
    0x1F011F011F010E01, 0x1F011F010E010E01, 0x01E001E001F101F1, 0xE001E001F101F101,

10.2.9.2.1  CryptSetOddByteParity()

This function sets the per byte parity of a 64-bit value. The least-significant bit is of each byte is replaced with the odd parity of the other 7 bits in the byte. With odd parity, no byte will ever be 0x00.

```c
UINT64 CryptSetOddByteParity(UINT64 k) {
    #define PMASK 0x0101010101010101ULL

    UINT64 out;
    k |= PMASK; // set the parity bit
    out = k;
    k ^= k >> 4; // odd parity extracted
    k ^= k >> 2;
    k ^= k >> 1;
    k &= PMASK; // out is now even parity because parity bit was already set
    out ^= k;
    return out;
}

10.2.9.2.2  CryptDesIsWeakKey()

Check to see if a DES key is on the list of weak, semi-weak, or possibly weak keys.
Return Value | Meaning
---|---
TRUE(1) | DES key is weak
FALSE(0) | DES key is not weak

```c
static BOOL CryptDesIsWeakKey(
    UINT64 k
) {
    int i;
    //
    for(i = 0; i < DES_NUM_WEAK; i++)
    {
        if(k == DesWeakKeys[i])
            return TRUE;
    }
    return FALSE;
}
```

### 10.2.9.2.3 CryptDesValidateKey()

Function to check to see if the input key is a valid DES key where the definition of valid is that none of the elements are on the list of weak, semi-weak, or possibly weak keys; and that for two keys, K1!=K2, and for three keys that K1!=K2 and K2!=K3.

```c
BOOL CryptDesValidateKey(
    TPM2B_SYM_KEY *desKey // IN: key to validate
) {
    UINT64 k[3];
    int i;
    int keys = (desKey->t.size + 7) / 8;
    BYTE *pk = desKey->t.buffer;
    BOOL ok;
    //
    // Note: 'keys' is the number of keys, not the maximum index for 'k'
    ok = ((keys == 2) || (keys == 3)) && ((desKey->t.size % 8) == 0);
    for(i = 0; ok && i < keys; pk += 8, i++)
    {
        k[i] = CryptSetOddByteParity(BYTE_ARRAY_TO_UINT64(pk));
        ok = !CryptDesIsWeakKey(k[i]);
    }
    ok = ok && k[0] != k[1];
    if(keys == 3)
        ok = ok && k[1] != k[2];
    return ok;
}
```

### 10.2.9.2.4 CryptGenerateKeyDes()

This function is used to create a DES key of the appropriate size. The key will have odd parity in the bytes.

```c
TPM_RC CryptGenerateKeyDes(
    TPMT_PUBLIC *publicArea, // IN/OUT: The public area template
    TPMT_SENSITIVE *sensitive, // OUT: sensitive area
    RAND_STATE *rand // IN: the "entropy" source for
```
81 )
82 {
83 // Assume that the publicArea key size has been validated and is a supported
84 // number of bits.
85 sensitive->sensitive.sym.t.size =
86     BITS_TO_BYTES(publicArea->parameters.symDetail.sym.keyBits.sym);
87 do
88     sensitive->sensitive.sym.t.size =
89     BITS_TO_BYTES(publicArea->parameters.symDetail.sym.keyBits.sym);
90     BYTE *pK = sensitive->sensitive.sym.t.buffer;
91     int i = (sensitive->sensitive.sym.t.size + 7) / 8;
92 // Use the random number generator to generate the required number of bits
93     if(DRDBG_Generate(rand, pK, sensitive->sensitive.sym.t.size) == 0)
94         return TPM_RC_NO_RESULT;
95     for(; i > 0; pK += 8, i--)
96         {
97             UINT64 k = BYTE_ARRAY_TO_UINT64(pK);
98             k = CryptSetOddByteParity(k);
99             UINT64_TO_BYTE_ARRAY(k, pK);
100         }
101     while(!CryptDesValidateKey(&sensitive->sensitive.sym));
102     return TPM_RC_SUCCESS;
103 }
104 #endif
10.2.10 CryptEccKeyExchange.c

10.2.10.1 Introduction

This file contains the functions that are used for the two-phase, ECC, key-exchange protocols.

```c
#include "Tpm.h"
#if CC_ZGen_2Phase == YES

10.2.10.2 Functions

#if ALG_ECMQV

10.2.10.2.1 avf1()

This function does the associated value computation required by MQV key exchange. Process:

a) Convert \( xQ \) to an integer \( xqi \) using the convention specified in Appendix C.3.
b) Calculate \( xqm = xqi \mod 2^\lceil f/2 \rceil \) (where \( f = \lceil \log_2(n) \rceil \)).
c) Calculate the associate value function \( avf(Q) = xqm + 2^{\lceil f / 2 \rceil} \) Always returns TRUE(1).

```c
static BOOL
avf1(
    bigNum               bnX,   // IN/OUT: the reduced value
    bigNum               bnN,   // IN: the order of the curve
)
{// compute \( f = 2^{\lceil \lceil \log_2(n) \rceil / 2 \rceil} \)
    int      f = (BnSizeInBits(bnN) + 1) / 2;
    // \( x' = 2^f + (x \mod 2^f) \)
    BnMaskBits(bnX, f); // This is mod 2*2^f but it doesn't matter because
    BnSetBit(bnX, f);  // the next operation will SET the extra bit anyway
    return TRUE;
}
```

10.2.10.2.2 C_2_2_MQV()

This function performs the key exchange defined in SP800-56A 6.1.1.4 Full MQV, C(2, 2, ECC MQV).

CAUTION: Implementation of this function may require use of essential claims in patents not owned by TCG members.

Points \( QsB() \) and \( QeB() \) are required to be on the curve of \( in\mathbf{QsA} \). The function will fail, possibly catastrophically, if this is not the case.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>the value for ( dsA ) does not give a valid point on the curve</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
C_2_2_MQV(
    TPM2B_ECC_PARAMETER *dsA, // IN: static private TPM key
    TPM2B_ECC_PARAMETER *deA, // IN: ephemeral private TPM key
    TPMMS_ECC_POINT *QsB,   // IN: static public party B key
    TPMMS_ECC_POINT *QeB,   // IN: ephemeral public party B key
    TPM2B_ECC_PARAMETER *dsB, // IN: static private TPM key
    TPM2B_ECC_PARAMETER *deB, // IN: ephemeral private TPM key
    TPMMS_ECC_POINT *QsA,   // IN: static public party A key
    TPMMS_ECC_POINT *QeA,   // IN: ephemeral public party A key
    TPMMS_ECC_POINT *QsC,   // IN: static public party C key
    TPMMS_ECC_POINT *QeC,   // IN: ephemeral public party C key
) { ...
```
```c
CURVE_INITIALIZED(E, curveId);
const ECC_CURVE_DATA    *C;
POINT(pQeA);
POINT_INITIALIZED(pQeB, QeB);
POINT_INITIALIZED(pQsB, QsB);
ECC_NUM(bnTa);
ECC_INITIALIZED(bnDeA, deA);
ECC_INITIALIZED(bnDsA, dsA);
ECC_NUM(bnN);
ECC_INITIALIZED(bnDeA, deA);
ECC_INITIALIZED(bnDsA, dsA);
ECC_NUM(bnN);
ECC_INITIALIZED(bnDeA, deA);
ECC_INITIALIZED(bnDsA, dsA);
ECC_NUM(bnN);
TPM_RC                 retVal;

// Parameter checks
if (E == NULL)
ERROR_RETURN(TPM_RC_VALUE);
pAssert(outZ != NULL && pQeB != NULL && pQsB != NULL && deA != NULL
&& dsA != NULL);
C = AccessCurveData(E);

// Process:
// 1. implicitsigA = (de,A + avf(Qe,A)ds,A ) mod n.
// 2. P = h(implicitsigA)(Qe,B + avf(Qe,B)Qs,B).
// 3. If P = O, output an error indicator.
// 4. Z=xP, where xP is the x-coordinate of P.

// Compute the public ephemeral key pQeA = [de,A]G
if((retVal = BnPointMult(pQeA, CurveGetG(C), bnDeA, NULL, NULL, E))
!= TPM_RC_SUCCESS)
goto Exit;

// 1. implicitsigA = (de,A + avf(Qe,A)ds,A ) mod n.
// tA := (ds,A + de,A  avf(Xe,A)) mod n    (3)
// Compute 'tA' = ('deA' + 'dsA'  avf('XeA')) mod n
// Ta = avf(XeA);
BnCopy(bnTa, pQeA->x);
avf1(bnTa, bnN);
// do Ta = ds,A * Ta mod n = dsA * avf(XeA) mod n
BnModMult(bnTa, bnDsA, bnTa, bnN);
// now Ta = deA + Ta mod n = deA + dsA * avf(XeA) mod n
BnAdd(bnTa, bnTa, bnDeA);
BnMod(bnTa, bnN);

// 2. P = h(implicitsigA)(Qe,B + avf(Qe,B)Qs,B).
// Put this in because almost every case of h is == 1 so skip the call when
// not necessary.
if(!BnEqualWord(CurveGetCofactor(C), 1))
    // Cofactor is not 1 so compute Ta := Ta * h mod n
    BnModMult(bnTa, bnTa, CurveGetCofactor(C), CurveGetOrder(C));
// Now that 'tA' is (h * 'tA' mod n)
// 'outZ' = (tA)(Qe,B + avf(Qe,B)Qs,B).
// first, compute XeB = avf(XeB)
avf1(bnXeB, bnN);

// QsB := [XeB]QsB
BnPointMult(pQsB, pQsB, bnXeB, NULL, NULL, E);
BnEccAdd(pQsB, pQsB, E);

// QeB := [tA]QeB = [tA](QsB + [Xe,B]QeB) and check for at infinity
// If the result is not the point at infinity, return QeB
BnPointMult(pQeB, pQeB, bnTa, NULL, NULL, E);
if(BnEqualZero(pQeB->x))
    ERROR_RETURN(TPM_RC_NO_RESULT);
// Convert BIGNUM E to TPM2B E
```

---

The code snippet above demonstrates the implementation of a protocol involving operations on elliptic curve cryptography (ECC). It is part of a larger library designed for trusted platform modules (TPMs), which are hardware components used to secure computer systems.

Here are the key points of the code:

1. **Initialization**: The code initializes various ECC-related data structures and constants, necessary for subsequent operations.

2. **Parameter Checks**: It includes checks for null pointers and other conditions to ensure the safety and validity of the input parameters.

3. **Process**:
   - **ImplicitsigA Calculation**: This involves the computation of a complex expression involving the ECC points and scalars.
   - **Z Computation**: After computing an intermediate result, it calculates the x-coordinate of a point, which is then used to compute the final output.

4. **Error Handling**: The code handles errors through error return codes, ensuring that any issues during the process are properly addressed.

This snippet is a part of a larger trusted platform module library, focusing on supporting routines for ECC operations, crucial for secure transactions and data protection in modern computing environments.
92     BnPointTo2B(outZ, pQeB, E);
93
94     Exit:
95     CURVE_FREE(E);
96     return retVal;
97
98 #endif // ALG_ECMQV

10.2.10.2.3 C_2_2_ECDH()

This function performs the two phase key exchange defined in SP800-56A, 6.1.1.2 Full Unified Model, C(2, 2, ECC CDH).

99 static TPM_RC
100 C_2_2_ECDH(
101     TPMS_ECC_POINT *outZs, // OUT: Zs
102     TPMS_ECC_POINT *outZe, // OUT: Ze
103     TPM2B_ECC_PARAMETER *dsA, // IN: static privat
104     TPM2B_ECC_PARAMETER *deA, // IN: ephemeral private TPM key
105     TPM_ECC_CURVE curveId, // IN: the curve for the computations
106     TPM_ECC_POINT *QsB, // IN: static public party B key
107     TPM_ECC_POINT *QeB // IN: ephemeral public party B key
108 )
109 {
110     CURVE_INITIALIZED(E, curveId);
111     ECC_INITIALIZED(bnAs, dsA);
112     ECC_INITIALIZED(bnAe, deA);
113     POINT_INITIALIZED(ecBs, QsB);
114     POINT_INITIALIZED(ecBe, QeB);
115     POINT(ecZ);
116     TPM_RC            retVal;
117     // Parameter checks
118     if(E == NULL)
119         ERROR_RETURN(TPM_RC_CURVE);
120     pAssert(outZs != NULL && dsA != NULL && deA != NULL && QsB != NULL && QeB != NULL);
121     // Do the point multiply for the Zs value ([dsA]QsB)
122     retVal = BnPointMult(ecZ, ecBs, bnAs, NULL, NULL, E);
123     if(retVal == TPM_RC_SUCCESS)
124         { // Convert the Zs value.
125             BnPointTo2B(outZs, ecZ, E);
126             // Do the point multiply for the Ze value ([deA]QeB)
127             retVal = BnPointMult(ecZ, ecBe, bnAe, NULL, NULL, E);
128             if(retVal == TPM_RC_SUCCESS)
129                 BnPointTo2B(outZe, ecZ, E);
130         }
131     Exit:
132     CURVE_FREE(E);
133     return retVal;
134 }

10.2.10.2.4 CryptEcc2PhaseKeyExchange()

This function is the dispatch routine for the EC key exchange functions that use two ephemeral and two static keys.
Error Returns | Meaning
--- | ---
TPM_RC_SCHEME | scheme is not defined

```
139 LIB_EXPORT TPM_RC
140 CryptEcc2PhaseKeyExchange(
141   TPMs_ECC_POINT *outZ1, // OUT: a computed point
142   TPMs_ECC_POINT *outZ2, // OUT: and optional second point
143   TPM ECC_CURVE curveId, // IN: the curve for the computations
144   TPM_ALG_ID scheme, // IN: the key exchange scheme
145   TPM2B ECC_PARAMETER *dsA, // IN: static private TPM key
146   TPM2B ECC_PARAMETER *deA, // IN: ephemeral private TPM key
147   TPMs_ECC_POINT *QsB, // IN: static public party B key
148   TPMs_ECC_POINT *QeB  // IN: ephemeral public party B key
149 )
150 {
151   pAssert(outZ1 != NULL
152     && dsA != NULL && deA != NULL
153     && QsB != NULL && QeB != NULL);
154
155   // Initialize the output points so that they are empty until one of the
156   // functions decides otherwise
157   outZ1->x.b.size = 0;
158   outZ1->y.b.size = 0;
159   if(outZ2 != NULL)
160   {
161     outZ2->x.b.size = 0;
162     outZ2->y.b.size = 0;
163   }
164   switch(scheme)
165   {
166     case ALG_ECDH_VALUE:
167       return C_2_2_ECDH(outZ1, outZ2, curveId, dsA, deA, QsB, QeB);
168     break;
169     #if ALG_ECMQV
170     case ALG_ECMQV VALUE:
171       return C_2_2_MQV(outZ1, curveId, dsA, deA, QsB, QeB);
172     break;
173     #endif
174     #if ALG_SM2
175     case ALG_SM2 VALUE:
176       return SM2KeyExchange(outZ1, curveId, dsA, deA, QsB, QeB);
177     break;
178     #endif
179   default:
180       return TPM_RC_SCHEME;
181   }
182 }
183 #if ALG_SM2

10.2.10.2.5 ComputeWForSM2()

Compute the value for w used by SM2

```
184 static UINT32
185 ComputeWForSM2 ( 
186   bigCurve    E
187 )
188 {
189   // w := ceil(ceil(log2(n)) / 2) - 1
190   return (BnMsb(CurveGetOrder(AccessCurveData(E))) / 2 - 1);
10.2.10.2.6  avfSm2()

This function does the associated value computation required by SM2 key exchange. This is different form the avf() in the international standards because it returns a value that is half the size of the value returned by the standard avf. For example, if \( n \) is 15, \( W_s \) (\( w \) in the standard) is 2 but the \( W \) here is 1. This means that an input value of 14 (1110b) would return a value of 110b with the standard but 10b with the scheme in SM2.

```c
static bigNum
avfSm2(bigNum              bn,    // IN/OUT: the reduced value
(UINT32              w       // IN: the value of \( w \)
)
{
    // a) set \( w := \text{ceil}(\text{ceil}(\log_2(n)) / 2) - 1 \)
    // b) set \( x' := 2^w + (x \& (2^w - 1)) \)
    // This is just like the avf for MQV where \( x' = 2^w + (x \mod 2^w) \)
    BnMaskBits(bn, w);    // as with avf1, this is too big by a factor of 2 but
    // it doesn't matter because we SET the extra bit
    BnSetBit(bn, w);
    return bn;
}
```

SM2KeyExchange() This function performs the key exchange defined in SM2. The first step is to compute \( t_A = (d_A + d_e \ avf(X_e, A)) \mod n \) Then, compute the Z value from \( \text{out}_Z = (h \ t_A \mod n) \ (Q_s A + \ avf(Q_e B).x) \ (Q_e B) \). The function will compute the ephemeral public key from the ephemeral private key. All points are required to be on the curve of \( \text{in}_Q s_A \). The function will fail catastrophically if this is not the case.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>the value for ( ds_A ) does not give a valid point on the curve</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT TPM_RC
SM2KeyExchange(
    TPM_band EC_POINT        *outZ,    // OUT: the computed point
    TPM_ECC_CURVE          curveId,    // IN: the curve for the computations
    TPM_ECC_PARAMETER   *dsAIn,       // IN: static private TPM key
    TPM_ECC_PARAMETER   *deAIn,       // IN: ephemeral private TPM key
    TPM_ECC_PARAMETER   *QsAIn,       // IN: static public party B key
    TPM_ECC_PARAMETER   *QeBIn        // IN: ephemeral public party B key
)
{
    CURVE_INITIALIZED(E, curveId);
    const ECC_CURVE_DATA      *C;
    ECC_INITIALIZED(dsA, dsAIn);
    ECC_INITIALIZED(deA, deAIn);
    POINT_INITIALIZED(QsB, QsBIn);
    POINT_INITIALIZED(QeB, QeBIn);
    BN_WORD_INITIALIZED(One, 1);
    POINT(QeA);
    ECC_NUM(XeB);
    POINT(Z);
    ECC_NUM(Ta);
    UINT32                   w;
    TPM_RC                   retVal = TPM_RC_NO_RESULT;
    // Parameter checks
    if(E == NULL)
        ERROR_RETURN(TPM_RC_CURVE);
    C = AccessCurveData(E);
```
236    pAssert(outZ != NULL && dsA != NULL && deA != NULL && QsB != NULL && QeB != NULL);
237
238    // Compute the value for w
239    w = ComputeWForSM2(E);
240
241    // Compute the public ephemeral key pQeA = [de,A]G
242    if(!BnEccModMult(QeA, CurveGetG(C), deA, E))
243        goto Exit;
244
245    // tA := (ds,A + de,A avf(Xe,A)) mod n  (3)
246    // Compute 'tA' = ('dsA' + 'deA' avf('XeA')) mod n
247    // Ta = avf(XeA);
248    // do Ta = de,A * Ta = deA * avf(XeA)
249    BnMult(Ta, deA, avfSm2(QeA->x, w));
250    // now Ta = dsA + Ta = dsA + deA * avf(XeA)
251    BnAdd(Ta, dsA, Ta);
252    BnMod(Ta, CurveGetOrder(C));
253
254    // outZ = [h  tA mod n] (Qs,B + [avf(Xe,B)](Qe,B)) (4)
255    // Put this in because almost every case of h is == 1 so skip the call when
256    // not necessary.
257    if(!BnEqualWord(CurveGetCofactor(C), 1))
258        // Cofactor is not 1 so compute Ta := Ta * h mod n
259        BnModMult(Ta, Ta, CurveGetCofactor(C), CurveGetOrder(C));
260    // Now that 'tA' is (h * 'tA' mod n)
261    // 'outZ' = ['tA'](QsB + [avf(QeB.x)](QeB)).
262    BnCopy(XeB, QeB->x);
263    if(!BnEccModMult2(Z, QsB, One, QeB, avfSm2(XeB, w), E))
264        goto Exit;
265    // QeB := [tA]QeB = [tA](QsB + [Xe,B]QeB) and check for at infinity
266    if(!BnEccModMult(Z, Z, Ta, E))
267        goto Exit;
268    // Convert BIGNUM E to TPM2B E
269    BnPointTo2B(outZ, Z, E);
270    retVal = TPM_RC_SUCCESS;
271    Exit:
272    CURVE_FREE(E);
273    return retVal;
274 }
275 #endif
276 #endif // CC_ZGen_2Phase
10.2.11 CryptEccMain.c

10.2.11.1 Includes and Defines

1 #include "Tpm.h"
2 #if ALG_ECC

This version requires that the new format for ECC data be used

3 #if !USE_BN_ECC_DATA
4 #error "Need to SET USE_BN_ECC_DATA to YES in Implementation.h"
5 #endif

10.2.11.2 Functions

6 #if SIMULATION
7 void
8 EccSimulationEnd(
9     void
10 )
11 {
12     #if SIMULATION
13     // put things to be printed at the end of the simulation here
14     #endif
15 }
16 #endif // SIMULATION

10.2.11.2.1 CryptEccInit()

This function is called at _TPM_Init()

17 BOOL
18 CryptEccInit(
19     void
20 )
21 {
22     return TRUE;
23 }

10.2.11.2.2 CryptEccStartup()

This function is called at TPM2_Startup().

24 BOOL
25 CryptEccStartup(
26     void
27 )
28 {
29     return TRUE;
30 }

10.2.11.2.3 ClearPoint2B(generic)

Initialize the size values of a TPMS_ECC_POINT structure.

31 void
32 ClearPoint2B(
33     TPMS_ECC_POINT *p        // IN: the point
34     } 
35     if(p != NULL) 
36     { 
37         p->x.t.size = 0; 
38         p->y.t.size = 0; 
39     } 
40 } 

10.2.11.2.4 CryptEccGetParametersByCurveId()

This function returns a pointer to the curve data that is associated with the indicated curveId. If there is no curve with the indicated ID, the function returns NULL. This function is in this module so that it can be called by GetCurve() data.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>curve with the indicated TPM_ECC_CURVE is not implemented</td>
</tr>
<tr>
<td>!= NULL</td>
<td>pointer to the curve data</td>
</tr>
</tbody>
</table>

LIB_EXPORT const ECC_CURVE *
CryptEccGetParametersByCurveId(
    TPM_ECC_CURVE       curveId       // IN: the curveID
)
{
    int i; 
    for(i = 0; i < ECC_CURVE_COUNT; i++)
    {
        if(eccCurves[i].curveId == curveId)
            return &eccCurves[i]; 
    }
    return NULL; 
}

10.2.11.2.5 CryptEccGetKeySizeForCurve()

This function returns the key size in bits of the indicated curve

LIB_EXPORT UINT16
CryptEccGetKeySizeForCurve(
    TPM_ECC_CURVE            curveId       // IN: the curve 
)
{
    const ECC_CURVE      *curve = CryptEccGetParametersByCurveId(curveId);
    UINT16           keySizeInBits; 
    //
    keySizeInBits = (curve != NULL) ? curve->keySizeBits : 0; 
    return keySizeInBits; 
}

10.2.11.2.6 GetCurveData()

This function returns the a pointer for the parameter data associated with a curve.

const ECC_CURVE_DATA *
GetCurveData(
    TPM_ECC_CURVE        curveId       // IN: the curveID
)
{
    const ECC_CURVE      *curve = CryptEccGetParametersByCurveId(curveId);
10.2.11.2.7 CryptEccGetCurveByIndex()

This function returns the number of the i-th implemented curve. The normal use would be to call this function with i starting at 0. When the i is greater than or equal to the number of implemented curves, TPM_ECC_NONE is returned.

```c
LIB_EXPORT TPM_ECC_CURVE
CryptEccGetCurveByIndex(
    UINT16               i
)
{
    if(i >= ECC_CURVE_COUNT)
        return TPM_ECC_NONE;
    return eccCurves[i].curveId;
}
```

10.2.11.2.8 CryptEccGetParameter()

This function returns an ECC curve parameter. The parameter is selected by a single character designator from the set of {PNABXYH}.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>curve exists and parameter returned</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>curve does not exist or parameter selector</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT BOOL
CryptEccGetParameter(
    TPM2B_ECC_PARAMETER     *out,     // OUT: place to put parameter
    char                     p,       // IN: the parameter selector
    TPM_ECC_CURVE            curveId  // IN: the curve id
)
{
    const ECC_CURVE_DATA    *curve = GetCurveData(curveId);
    bigConst                 parameter = NULL;

    if(curve != NULL)
        {
            switch(p)
            {
                case 'p':
                    parameter = CurveGetPrime(curve);
                    break;
                case 'n':
                    parameter = CurveGetOrder(curve);
                    break;
                case 'a':
                    parameter = CurveGet_a(curve);
                    break;
                case 'b':
                    parameter = CurveGet_b(curve);
                    break;
                case 'x':
                    parameter = CurveGetGx(curve);
                    break;
                case 'y':
                    parameter = CurveGetGy(curve);
                    break;
            }
        }
    return parameter;  
} 
```
case 'h':
    parameter = CurveGetCofactor(curve);
    break;
default:
    FAIL(FATAL_ERROR_INTERNAL);
    break;
}

// If not debugging and we get here with parameter still NULL, had better
// not try to convert so just return FALSE instead.
return (parameter != NULL) ? BnTo2B(parameter, &out->b, 0) : 0;

10.2.11.2.9 CryptCapGetECCCurve()

This function returns the list of implemented ECC curves.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>if no more ECC curve is available</td>
</tr>
<tr>
<td>NO</td>
<td>if there are more ECC curves not reported</td>
</tr>
</tbody>
</table>

TPMI_YES_NO

CryptCapGetECCCurve(
    TPM_ECC_CURVE curveID,     // IN: the starting ECC curve
    UINT32 maxCount,          // IN: count of returned curves
    TPML_ECC_CURVE *curveList // OUT: ECC curve list
)
{
    TPMI_YES_NO more = NO;
    UINT16 i;
    UINT32 count = ECC_CURVE_COUNT;
    TPM_ECC_CURVE curve;

    // Initialize output property list
    curveList->count = 0;

    // The maximum count of curves we may return is MAX_ECC_CURVES
    if(maxCount > MAX_ECC_CURVES) maxCount = MAX_ECC_CURVES;

    // Scan the eccCurveValues array
    for(i = 0; i < count; i++)
        {
            curve = CryptEccGetCurveByIndex(i);
            // If curveID is less than the starting curveID, skip it
            if(curve < curveID)
                continue;
            if(curveList->count < maxCount)
                {
                    // If we have not filled up the return list, add more curves to
                    // it
                    curveList->eccCurves[curveList->count] = curve;
                    curveList->count++;
                }
            else
                {
                    // If the return list is full but we still have curves
                    // available, report this and stop iterating
                    more = YES;
                    break;
                }
        }
    return more;
}
10.2.11.2.10 CryptGetCurveSignScheme()

This function will return a pointer to the scheme of the curve.

```c
const TPMT_ECC_SCHEME *
CryptGetCurveSignScheme(
    TPM_ECC_CURVE    curveId); // IN: The curve selector
{  
    const ECC_CURVE         *curve = CryptEccGetParametersByCurveId(curveId);
    if(curve != NULL) return &(curve->sign);
    else return NULL;
}
```

10.2.11.2.11 CryptGenerateR()

This function computes the commit random value for a split signing scheme.

If `c` is NULL, it indicates that `r` is being generated for `TPM2_Commit()`. If `c` is not NULL, the TPM will validate that the `gr.commitArray` bit associated with the input value of `c` is SET. If not, the TPM returns FALSE and no `r` value is generated.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td><code>r</code> value computed</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>no <code>r</code> value computed</td>
</tr>
</tbody>
</table>

```c
BOOL
CryptGenerateR(
    TPM2B_ECC_PARAMETER     *r,    // OUT: the generated random value
    UINT16                  *c,     // IN/OUT: count value.
    TPMI_ECC_CURVE           curveID, // IN: the curve for the value
    TPM2B_NAME              *name   // IN: optional name of a key to
    // associate with 'r'
)
{  
    // This holds the marshaled g_commitCounter.
    TPM2B_TYPE(8B, 8);
    TPM2B_8B                 cntr = {{8,{0}}};
    UINT32                   iterations;
    TPM2B_ECC_PARAMETER      n;
    UINT64                   currentCount = gr.commitCounter;
    UINT16                   t1;
    //
    if(!CryptEccGetParameter(&n, 'n', curveID))
        return FALSE;
    // If this is the commit phase, use the current value of the commit counter
    if(c != NULL) {
        // if the array bit is not set, can't use the value.
        if(!TEST_BIT(*c & COMMIT_INDEX_MASK, gr.commitArray))
            return FALSE;
        // If it is the sign phase, figure out what the counter value was
        // when the commitment was made.
        //
```
// When gr.commitArray has less than 64K bits, the extra
// bits of 'c' are used as a check to make sure that the
// signing operation is not using an out of range count value
t1 = (UINT16)currentCount;

// If the lower bits of c are greater or equal to the lower bits of t1
// then the upper bits of t1 must be one more than the upper bits
// of c
if((c & COMMIT_INDEX_MASK) >= (t1 & COMMIT_INDEX_MASK))
    currentCount = currentCount - (COMMIT_INDEX_MASK + 1);

// Marshal the count value to a TPM2B buffer for the KDF
cntr.t.size = sizeof(currentCount);
UINT64_TO_BYTE_ARRAY(currentCount, cntr.t.buffer);

// Now can do the KDF to create the random value for the signing operation
// During the creation process, we may generate an r that does not meet the
// requirements of the random value.
// want to generate a new r.
for(iterations = 1; iterations < 1000000;)
{
    int i;
    CryptKDFa(CONTEXT_INTEGRITY_HASH_ALG, &gr.commitNonce.b, COMMIT_STRING,
             &name->b, &cntr.b, n.t.size * 8, r->t.buffer, &iterations, FALSE);

    // "random" value must be less than the prime
    if(UnsignedCompareB(r->b.size, r->b.buffer, n.t.size, n.t.buffer) >= 0)
        continue;

    // in this implementation it is required that at least bit
    // in the upper half of the number be set
    for(i = n.t.size / 2; i >= 0; i--)
        if(r->b.buffer[i] != 0)
            return TRUE;

    return FALSE;
}

10.2.11.2.12 CryptCommit()

This function is called when the count value is committed. The gr.commitArray value associated with the
current count value is SET and g_commitCounter is incremented. The low-order 16 bits of old value of the
counter is returned.

UINT16 CryptCommit(
    void
)
{
    UINT16 oldCount = (UINT16)gr.commitCounter;
gr.commitCounter++;
    SET_BIT(oldCount & COMMIT_INDEX_MASK, gr.commitArray);
    return oldCount;
}
10.2.11.2.13 CryptEndCommit()

This function is called when the signing operation using the committed value is completed. It clears the gr.commitArray bit associated with the count value so that it can't be used again.

```c
268 void CryptEndCommit(  
269     UINT16 c // IN: the counter value of the commitment  
270 )  
271 {  
272     ClearBit((c & COMMIT_INDEX_MASK), gr.commitArray, sizeof(gr.commitArray));  
273 }
```

10.2.11.2.14 CryptEccGetParameters()

This function returns the ECC parameter details of the given curve

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>unsupported ECC curve ID</td>
</tr>
</tbody>
</table>

```c
275 BOOL CryptEccGetParameters(  
276     TPM_ECC_CURVE curveId, // IN: ECC curve ID  
277     TPMS_ALGORITHM_DETAIL_ECC *parameters // OUT: ECC parameters  
278 )  
279 {  
280     const ECC_CURVE *curve = CryptEccGetParametersByCurveId(curveId);  
281     const ECC_CURVE_DATA *data;  
282     BOOL found = curve != NULL;  
283     if(found)  
284     {  
285         data = curve->curveData;  
286         parameters->keySize = curve->keySizeBits;  
287         parameters->kdf = curve->kdf;  
288         parameters->sign = curve->sign;  
289         BnTo2B(data->p, &parameters->p.b, 0);  
290         BnTo2B(data->a, &parameters->a.b, 0);  
291         BnTo2B(data->b, &parameters->b.b, 0);  
292         BnTo2B(data->gX, &parameters->gX.b, parameters->p.t.size);  
293         BnTo2B(data->gY, &parameters->gY.b, parameters->p.t.size);  
294         BnTo2B(data->order, &parameters->n.b, 0);  
295         BnTo2B(data->h, &parameters->h.b, 0);  
296     }  
297     return found;  
298 }
```

10.2.11.2.15 BnGetCurvePrime()

This function is used to get just the prime modulus associated with a curve

```c
304 const bignum_t * BnGetCurvePrime(  
305     TPM_ECC_CURVE curveId  
306 )  
307 {
```
const ECC_CURVE_DATA *C = GetCurveData(curveId);
return (C != NULL) ? CurveGetPrime(C) : NULL;
}

10.2.11.2.16 BnGetCurveOrder()

This function is used to get just the curve order.

const bignum_t *
BnGetCurveOrder(
TPM_ECC_CURVE curveId
)
{
const ECC_CURVE_DATA *C = GetCurveData(curveId);
return (C != NULL) ? CurveGetOrder(C) : NULL;
}

10.2.11.2.17 BnIsOnCurve()

This function checks if a point is on the curve.

BOOL
BnIsOnCurve(
pointConst Q,
const ECC_CURVE_DATA *C
)
{
BN_VAR(right, (MAX_ECC_KEY_BITS * 3));
BN_VAR(left, (MAX_ECC_KEY_BITS * 2));
bigConst prime = CurveGetPrime(C);
// Show that point is on the curve y^2 = x^3 + ax + b;
// Or y^2 = x(x^2 + a) + b
// y^2
BnMult(left, Q->y, Q->y);
BnMod(left, prime);
// x^2
BnMult(right, Q->x, Q->x);
// x^2 + a
BnAdd(right, right, CurveGet_a(C));
// BnMod(right, CurveGetPrime(C));
// x(x^2 + a)
BnMult(right, right, Q->x);
// x(x^2 + a) + b
BnAdd(right, right, CurveGet_b(C));
BnMod(right, prime);
if (BnUnsignedCmp(left, right) == 0)
    return TRUE;
else
    return FALSE;
}

10.2.11.2.18 BnIsValidPrivateEcc()

Checks that 0 < x < q
BOOL
BnIsValidPrivateEcc(
    bigConst x,       // IN: private key to check
    bigCurve E       // IN: the curve to check
)
{
    BOOL        retVal;
    retVal = (!BnEqualZero(x)
                && (BnUnsignedCmp(x, CurveGetOrder(AccessCurveData(E))) < 0));
    return retVal;
}

LIB_EXPORT BOOL
CryptEccIsValidPrivateKey(
    TPM2B_ECC_PARAMETER     *d,
    TPM_ECC_CURVE            curveId
)
{
    BN_INITIALIZED(bnD, MAX_ECC_PARAMETER_BYTES * 8, d);
    return !BnEqualZero(bnD) && (BnUnsignedCmp(bnD, BnGetCurveOrder(curveId)) < 0);
}

10.2.11.2.19 BnPoinPointMult()

This function does a point multiply of the form R = [d]S + [u]Q where the parameters are bigNum values. If S is NULL and d is not NULL, then it computes R = [d]G + [u]Q or just R = [d]G if u and Q are NULL. If skipChecks is TRUE, then the function will not verify that the inputs are correct for the domain. This would be the case when the values were created by the CryptoEngine() code. It will return TPM_RC_NO_RESULT if the resulting point is the point at infinity.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>result of multiplication is a point at infinity</td>
</tr>
<tr>
<td>TPM_RC_ECC_POINT</td>
<td>S or Q is not on the curve</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>d or u is not &lt; n</td>
</tr>
</tbody>
</table>

BOOL
BnPoinPointMult(
    bigPoint R,       // OUT: computed point
    pointConst S,     // IN: optional point to multiply by 'd'
    bigConst d,       // IN: scalar for [d]S or [d]G
    pointConst Q,     // IN: optional second point
    bigConst u,       // IN: optional second scalar
    bigCurve E        // IN: curve parameters
)
{
    BOOL                 OK;
    //
    TEST(TPM_ALG_ECDH);

    // Need one scalar
    OK = (d != NULL || u != NULL);
    if (!OK)
    {
        return TPM_RC_NO_RESULT;
    }
    // If S is present, then d has to be present. If S is not
    // present, then d may or may not be present
    OK = OK && (((S == NULL) == (d == NULL)) || (d != NULL));
    // either both u and Q have to be provided or neither can be provided (don't
    // know what to do if only one is provided.
    OK = OK && ((u == NULL) == (Q == NULL));
    if (!OK)
    {
        return TPM_RC_NO_RESULT;
    }
    return OK && (E != NULL);
return TPM_RC_VALUE;

OK = (S == NULL) || BnIsOnCurve(S, AccessCurveData(E));
OK = OK && ((Q == NULL) || BnIsOnCurve(Q, AccessCurveData(E)));
if(!OK)
    return TPM_RC_ECC_POINT;

if((d != NULL) && (S == NULL))
    S = CurveGetG(AccessCurveData(E));
// If only one scalar, don't need Shamir's trick
if((d == NULL) || (u == NULL))
{
    if(d == NULL)
        OK = BnEccModMult(R, Q, u, E);
    else
        OK = BnEccModMult(R, S, d, E);
}
else
{
    OK = BnEccModMult2(R, S, d, Q, u, E);
}
return (OK ? TPM_RC_SUCCESS : TPM_RC_NO_RESULT);

10.2.11.2.20 BnEccGetPrivate()

This function gets random values that are the size of the key plus 64 bits. The value is reduced (mod (q - 1)) and incremented by 1 (q is the order of the curve. This produces a value (d) such that 1 <= d < q. This is the method of FIPS 186-4 Section B.4.1 'Key Pair Generation Using Extra Random Bits'.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure generating private key</td>
</tr>
</tbody>
</table>

BOOL BnEccGetPrivate(
    bigNum dOut,   // OUT: the qualified random value
    const ECC_CURVE_DATA *C,  // IN: curve for which the private key
    RAND_STATE *rand // IN: state for DRBG
)
{
    bigConst order = CurveGetOrder(C);
    BOOL OK;
    UINT32 orderBits = BnSizeInBits(order);
    UINT32 orderBytes = BITS_TO_BYTES(orderBits);
    BN_VAR(bnExtraBits, MAX_ECC_KEY_BITS + 64);
    BN_VAR(nMinus1, MAX_ECC_KEY_BITS);
    //
    OK = BnGetRandomBits(bnExtraBits, (orderBytes * 8) + 64, rand);
    OK = OK && BnSubWord(nMinus1, order, 1);
    OK = OK && BnMod(bnExtraBits, nMinus1);
    OK = OK && BnAddWord(dOut, bnExtraBits, 1);
    return OK;
}

10.2.11.2.21 BnEccGenerateKeyPair()

This function gets a private scalar from the source of random bits and does the point multiply to get the public key.
10.2.11.2.22 CryptEccNewKeyPair

This function creates an ephemeral ECC. It is ephemeral in that it is expected that the private part of the key will be discarded.

```c
LIB_EXPORT TPM_RC
CryptEccNewKeyPair(
  TPMS_ECC_POINT *Qout,    // OUT: the public point
  TPM2B_ECC_PARAMETER *dOut,  // OUT: the private scalar
  TPM_ECC_CURVE curveId    // IN: the curve for the key
)
{
  CURVE_INITIALIZED(E, curveId);
  POINT(ecQ);
  ECC_NUM(bnD);
  BOOL OK;

  if(E == NULL)
  {
    return TPM_RC_CURVE;
  }

  TEST(TPM_ALG_ECDH);
  OK = BnEccGenerateKeyPair(bnD, ecQ, E, NULL);
  if(OK)
  {
    BnPPointTo2B(Qout, ecQ, E);
    BnPTo2B(bnD, &dOut->b, Qout->x.t.size);
  }
  else
  {
    Qout->x.t.size = Qout->y.t.size = dOut->t.size = 0;
  }
  CURVE_FREE(E);
  return OK ? TPM_RC_SUCCESS : TPM_RC_NO_RESULT;
}
```

10.2.11.2.23 CryptEccPointMultiply()

This function computes \( R := [dln]G + [uln]Qln \). Where \( dln \) and \( uln \) are scalars, \( G \) and \( Qln \) are points on the specified curve and \( G \) is the default generator of the curve.

The \( xOut \) and \( yOut \) parameters are optional and may be set to NULL if not used.
It is not necessary to provide \( uIn \) if \( QIn \) is specified but one of \( uIn \) and \( dIn \) must be provided. If \( dIn \) and \( QIn \) are specified but \( uIn \) is not provided, then \( R = [dIn]QIn \).

If the multiply produces the point at infinity, the TPM_RC_NO_RESULT is returned.

The sizes of \( xOut \) and \( yOut \) will be set to be the size of the degree of the curve.

It is a fatal error if \( dIn \) and \( uIn \) are both unspecified (NULL) or if \( Qin \) or \( Rout \) is unspecified.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ECC_POINT</td>
<td>the point ( Pin ) or ( Qin ) is not on the curve</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>the product point is at infinity</td>
</tr>
<tr>
<td>TPM_RC_CURVE</td>
<td>bad curve</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>( dIn ) or ( uIn ) out of range</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT TPM_RC
CryptEccPointMultiply(
    TPMS_ECC_POINT *Rout, // OUT: the product point R
    TPM_ECC_CURVE curveId, // IN: the curve to use
    TPMS_ECC_POINT *Pin,   // IN: first point (can be null)
    TPM2B_ECC_PARAMETER *dIn, // IN: scalar value for \([dIn]Qin\)
    TPMS_ECC_POINT *Qin,   // IN: point Q
    TPM2B_ECC_PARAMETER *uIn // IN: scalar value for the multiplier of Q
) {
    CURVE_INITIALIZED(E, curveId);
    POINT_INITIALIZED(ecP, Pin);
    ECC_INITIALIZED(bnD, dIn);   // If dIn is null, then bnD is null
    ECC_INITIALIZED(bnU, uIn);
    POINT_INITIALIZED(ecQ, Qin);
    POINT(ecR);
    TPM_RC retVal;
    // retVal = BnPointMult(ecR, ecP, bnD, ecQ, bnU, E);
    if(retVal == TPM_RC_SUCCESS)
        BnPointTo2B(Rout, ecR, E);
    else
        ClearPoint2B(Rout);
    CURVE_FREE(E);
    return retVal;
}
```

### 10.2.11.2.24 CryptEccIsPointOnCurve()

This function is used to test if a point is on a defined curve. It does this by checking that \( y^2 \mod p = x^3 + a \cdot x + b \mod p \)

It is a fatal error if \( Q \) is not specified (is NULL).

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>point is on curve</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>point is not on curve or curve is not supported</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT BOOL
CryptEccIsPointOnCurve(
    TPM_ECC_CURVE curveId,   // IN: the curve selector
) {
    // Check if point is on curve
    if(condition) return TRUE;
    else return FALSE;
}
```
10.2.11.2.25 CryptEccGenerateKey()

This function generates an ECC key pair based on the input parameters. This routine uses KDFa() to produce candidate numbers. The method is according to FIPS 186-3, section B.1.2 "Key Pair Generation by Testing Candidates." According to the method in FIPS 186-3, the resulting private value \( d \) should be \( 1 \leq d < n \) where \( n \) is the order of the base point.

It is a fatal error if \( Qout, dOut \), is not provided (is NULL).

If the curve is not supported If \( seed \) is not provided, then a random number will be used for the key.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CURVE</td>
<td>curve is not supported</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>could not verify key with signature (FIPS only)</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT TPM_RC
CryptEccGenerateKey(
  TPMT_PUBLIC  *publicArea, // IN/OUT: The public area template for
                // the new key. The public key
                // area will be replaced computed
                // ECC public key
  TPMT_SENSITIVE  *sensitive, // OUT: the sensitive area will be
                    // updated to contain the private
                    // ECC key and the symmetric
                    // encryption key
  RAND_STATE      *rand    // IN: if not NULL, the deterministic
                    // RNG state
)
{ // TEST(TPM_ALG_ECDSA); // ECDSA is used to verify each key
  CURVE_INITIALIZED(E, publicArea->parameters.eccDetail.curveID);
  ECC_NUM(bnD);
  POINT(ecQ);
  BOOL          OK;
  TPM_RC        retVal;
  // Validate parameters
  if(E == NULL)
    ERROR_RETURN(TPM_RC_CURVE);
  publicArea->unique.ecc.x.t.size = 0;
  publicArea->unique.ecc.y.t.size = 0;
  sensitive->sensitive.ecc.t.size = 0;
  OK = BnEccGenerateKeyPair(bnD, ecQ, E, rand);
  if(OK)
  {
    BnPointTo2B(&publicArea->unique.ecc, ecQ, E);
    BnTo2B(bnD, &sensitive->sensitive.ecc.b, publicArea->unique.ecc.x.t.size);
  }
```
#if FIPS_COMPLIANT
   // See if PWCT is required
   if(OK && IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
   {
      ECC_NUM(bnT);
      ECC_NUM(bnS);
      TPM2B_DIGEST digest;
      // TEST(TPM_ALG_ECSA);
      digest.t.size = MIN(sensitive->sensitive.ecc.t.size, sizeof(digest.t.buffer));
      // Get a random value to sign using the built in DRBG state
      DRBG_Generate(NULL, digest.t.buffer, digest.t.size);
      BnSignEcdsa(bnT, bnS, E, bnD, &digest, NULL);
      // and make sure that we can validate the signature
      OK = BnValidateSignatureEcdsa(bnT, bnS, E, ecQ, &digest) == TPM_RC_SUCCESS;
   }
#endif
retVal = (OK) ? TPM_RC_SUCCESS : TPM_RC_NO_RESULT;
Exit:
   CURVE_FREE(E);
   return retVal;
#endif // ALG_ECC
10.2.12 CryptEccSignature.c

10.2.12.1 Includes and Defines

```c
#include "Tpm.h"
#include "CryptEccSignature_fp.h"
#if ALG_ECC
```

10.2.12.2 Utility Functions

10.2.12.2.1 EcdsaDigest()

Function to adjust the digest so that it is no larger than the order of the curve. This is used for ECDSA sign and verification.

```c
static bigNum EcdsaDigest(
    bigNum          bnD,          // OUT: the adjusted digest
    const TPM2B_DIGEST *digest,  // IN: digest to adjust
    bigConst        max           // IN: value that indicates the maximum
                                  // number of bits in the results
)
{
    int bitsInMax = BnSizeInBits(max);
    int shift;
    //
    if(digest == NULL)
        BnSetWord(bnD, 0);
    else
    {
        BnFromBytes(bnD, digest->t.buffer,
                    (NUMBYTES)MIN(digest->t.size, BITS_TO_BYTES(bitsInMax)));
        shift = BnSizeInBits(bnD) - bitsInMax;
        if(shift > 0)
            BnShiftRight(bnD, bnD, shift);
    }
    return bnD;
}
```

10.2.12.2.2 BnSchnorrSign()

This contains the Schnorr signature computation. It is used by both ECDSA and Schnorr signing. The result is computed as: \[ s = k + r \cdot d \mod n \] where

a) \( s \) is the signature  
b) \( k \) is a random value  
c) \( r \) is the value to sign  
d) \( d \) is the private EC key  
e) \( n \) is the order of the curve

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>the result of the operation was zero or ( r \mod n ) is zero</td>
</tr>
</tbody>
</table>

```c
static TPM_RC BnSchnorrSign(
    bigNum          bnS,          // OUT: s component of the signature
)```

```c
```
```c
```
30 bigConst bnK,  // IN: a random value
31 bigNum bnR,   // IN: the signature 'r' value
32 bigConst bnD,  // IN: the private key
33 bigConst bnN   // IN: the order of the curve
34 }
35 {
36 // Need a local temp value to store the intermediate computation because product
37 // size can be larger than will fit in bnS.
38 BN_VAR(bnT1, MAX_ECC_PARAMETER_BYTES * 2 * 8);
39 //
40 // Reduce bnR without changing the input value
41 BnDiv(NULL, bnT1, bnR, bnN);
42 if(BnEqualZero(bnT1))
43 return TPM_RC_NO_RESULT;
44 // compute s = (k + r * d)(mod n)
45 // r * d
46 BnMult(bnT1, bnT1, bnD);
47 // k * r * d
48 BnAdd(bnT1, bnT1, bnK);
49 // k + r * d (mod n)
50 BnDiv(NULL, bnS, bnT1, bnN);
51 return (BnEqualZero(bnS)) ? TPM_RC_NO_RESULT : TPM_RC_SUCCESS;
52 }

10.2.12.3 Signing Functions

10.2.12.3.1 BnSignEcdsa()

This function implements the ECDSA signing algorithm. The method is described in the comments below. This version works with internal numbers.

TPM_RC
BnSignEcdsa(
bigNum bnR,   // OUT: r component of the signature
bigNum bnS,   // OUT: s component of the signature
bigCurve E,   // IN: the curve used in the signature
bigNum bnD,   // IN: private signing key
const TPM2B_DIGEST *digest,  // IN: the digest to sign
RAND_STATE *rand     // IN: used in debug of signing
)
{
ECC_NUM(bnK);
ECC_NUM(bnIk);
BN_VAR(bnE, MAX(MAX_ECC_KEY_BYTES, MAX_DIGEST_SIZE) * 8);
POINT(ecR);
bigConst order = CurveGetOrder(AccessCurveData(E));
TPM_RC retVal = TPM_RC_SUCCESS;
INT32 tries = 10;
BOOL OK = FALSE;
// pAssert(digest != NULL);
// The algorithm as described in "Suite B Implementer's Guide to FIPS
// 186-3(ECDSA)"
// 1. Use one of the routines in Appendix A.2 to generate (k, k^-1), a
// per-message secret number and its inverse modulo n. Since n is prime,
// the output will be invalid only if there is a failure in the RBG.
// 2. Compute the elliptic curve point R = [k]G = (xR, yR) using EC scalar
// multiplication (see [Routines]), where G is the base point included in
// the set of domain parameters.
// 3. Compute r = xR mod n. If r = 0, then return to Step 1. 1.
// 4. Use the selected hash function to compute H = Hash(M).
// 5. Convert the bit string H to an integer e as described in Appendix B.2.
// 6. Compute \( s = (k^{n-1} \cdot (e + d \cdot r)) \mod q \). If \( s = 0 \), return to Step 1.2.
// 7. Return \((r, s)\).
// In the code below, \( q \) is \( n \) (that it, the order of the curve is \( p \))
do // This implements the loop at step 6. If \( s \) is zero, start over.
{
    for (; tries > 0; tries--)
    {
        // Step 1 and 2 -- generate an ephemeral key and the modular inverse
        // of the private key.
        if (!BnEccGenerateKeyPair(bnK, ecR, E, rand))
            continue;
        // \( x \) coordinate is mod \( p \). Make it mod \( q \)
        BnMod(ecR->x, order);
        // Make sure that it is not zero;
        if (BnEqualZero(ecR->x))
            continue;
        // write the modular reduced version of \( r \) as part of the signature
        BnCopy(bnR, ecR->x);
        // Make sure that a modular inverse exists and try again if not
        OK = (BnModInverse(bnIk, bnK, order));
        if (!OK)
            break;
    }
    if (!OK)
        goto Exit;
    EcdsaDigest(bnE, digest, order);
    // now have inverse of \( K \) (bnIk), \( e \) (bnE), \( r \) (bnR), \( d \) (bnD) and
    // CurveGetOrder(E)
    // Compute \( s = k^{n-1} \cdot (e + r \cdot d) \mod q \)
    // first do \( s = r \cdot d \mod q \)
    BnModMult(bnS, bnR, bnD, order);
    // \( s = e + s = e + r \cdot d \)
    BnAdd(bnS, bnE, bnS);
    // \( s = k^{n-1} \cdot s \mod n = k^{n-1} \cdot (e + r \cdot d) \mod n \)
    BnModMult(bnS, bnIk, bnS, order);
    // If \( S \) is zero, try again
} while (BnEqualZero(bnS));
Exit: return retVal;
}
#endif

10.2.12.3.2 BnSignEcdaa()

This function performs \( s = r + T \cdot d \mod q \) where

\(^{(a)}\) \( r \) is a random, or pseudo-random value created in the commit phase
\(^{(b)}\) \( nonceK \) is a TPM-generated, random value \( 0 < nonceK < n \)
\(^{(c)}\) \( T \) is mod \( q \) of Hash(\( nonceK || digest \)), and
\(^{(d)}\) \( d \) is a private key.

The signature is the tuple \((nonceK, s)\)

Regrettably, the parameters in this function kind of collide with the parameter
### Error Returns

<table>
<thead>
<tr>
<th>Error Return</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SCHEME</td>
<td>unsupported hash algorithm</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>cannot get values from random number generator</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
BnSignEcdaa(
    TPM2B_ECC_PARAMETER *nonceK,     // OUT: nonce component of the signature
    bigNum bnS,                     // OUT: s component of the signature
    bigCurve E,                     // IN: the curve used in signing
    bigNum bnD,                     // IN: the private key
    const TPM2B_DIGEST *digest,     // IN: the value to sign (mod q)
    TPM2B_ECC_PARAMETER *scheme,    // IN: signing scheme (contains the
                                      // commit count value).
    OBJECT *eccKey,                 // IN: The signing key
    RAND_STATE *rand                // IN: a random number state
)
{
    TPM_RC retVal;
    TPM2B_ECC_PARAMETER r;
    HASH_STATE state;
    TPM2B_DIGEST T;
    BN_MAX(bnT);
    // NOT REFERENCED(rand);
    if(!CryptGenerateR(&r, &scheme->details.ecdaa.count,
                       eccKey->publicArea.parameters.eccDetail.curveID,
                       eccKey->name))
        retVal = TPM_RC_VALUE;
    else
    {
        // This allocation is here because 'r' doesn't have a value until
        // CryptGenerateR() is done.
        ECC_INITIALIZED(bnR, &r);
        do
        {
            // generate nonceK such that 0 < nonceK < n
            // use bnT as a temp.
            if(!BnEccGetPrivate(bnT, AccessCurveData(E), rand))
                { retVal = TPM_RC_NO_RESULT;
                break;
                }
            BnTo2B(bnT, &nonceK->b, 0);
            T.t.size = CryptHashStart(&state, scheme->details.ecdaa.hashAlg);
            if(T.t.size == 0)
                { retVal = TPM_RC_SCHEME;
                }
            else
                { CryptDigestUpdate2B(&state, &nonceK->b);
                CryptDigestUpdate2B(&state, &digest->b);
                CryptHashEnd2B(&state, &T.b);
                BnFrom2B(bnT, &T.b);
                // Watch out for the name collisions in this call!!
                retVal = BnSchnorrSign(bnS, bnR, bnT, bnD,
                                        AccessCurveData(E)->order);
                }
        } while(retVal == TPM_RC_NO_RESULT);
    // Because the rule is that internal state is not modified if the command
    // fails, only end the commit if the command succeeds.
    // NOTE that if the result of the Schnorr computation was zero
```
189  // it will probably not be worthwhile to run the same command again because
190  // the result will still be zero. This means that the Commit command will
191  // need to be run again to get a new commit value for the signature.
192  if (retVal == TPM_RC_SUCCESS)
193      CryptEndCommit(scheme->details.ecda.count);
194  }
195  return retVal;
196 }
197 #endif // ALG_ECDAA
198 #if ALG_ECSCHNORR

### 10.2.12.3 SchnorrReduce()

Function to reduce a hash result if it's magnitude is too large. The size of number is set so that it has no more bytes of significance than the reference value. If the resulting number can have more bits of significance than the reference.

```c
199 static void
200 SchnorrReduce(
201  TPM2B       *number,  // IN/OUT: Value to reduce
202  bigConst     reference  // IN: the reference value
203 )
204 {
205  UINT16 maxBytes = (UINT16)BITS_TO_BYTES(BnSizeInBits(reference));
206  if (number->size > maxBytes)
207       number->size = maxBytes;
208 }
```

### 10.2.12.3.4 SchnorrEcc()

This function is used to perform a modified Schnorr signature. This function will generate a random value k and compute

a) \((xR, yR) = [k]G\)
b) \(r = \text{hash}(xR || P)(\text{mod } q)\)
c) \(rT = \text{truncated } r\)
d) \(s = k + rT * ds \pmod{q}\)
e) return the tuple \(rT, s\)

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>failure in the Schnorr sign process</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>hashAlg can't produce zero-length digest</td>
</tr>
</tbody>
</table>

```c
209 static TPM_RC
210 BnSignEcSchnorr(
211  bigNum       bnR,  // OUT: r component of the signature
212  bigNum       bnS,  // OUT: s component of the signature
213  bigCurve     E,   // IN: the curve used in signing
214  bigNum       bnD,  // IN: the signing key
215  const TPM2B_DIGEST *digest, // IN: the digest to sign
216  TPM_ALG_ID   hashAlg, // IN: signing scheme (contains a hash)
217  RAND_STATE   *rand    // IN: non-NULL when testing
218 )
219 {
220  HASH_STATE hashState;
221  UINT16     digestSize = CryptHashGetDigestSize(hashAlg);
222  TPM2B_TYPE(T, MAX(MAX_DIGEST_SIZE, MAX_ECC_KEY_BYTES));
```
TPM2B_T T2b;
TPM2B *e = &T2b.b;

TPM_RC retVal = TPM_RC_NO_RESULT;

const ECC_CURVE_DATA *C;
bigConst order;
bigConst prime;
ECC_NUM(bnK);
POINT(ecR);

// Parameter checks
if (E == NULL)
   ERROR_RETURN(TPM_RC_VALUE);
C = AccessCurveData(E);
order = CurveGetOrder(C);
prime = CurveGetOrder(C);

// If the digest does not produce a hash, then null the signature and return
// a failure.
if (digestSize == 0)
{
   BnSetWord(bnR, 0);
   BnSetWord(bnS, 0);
   ERROR_RETURN(TPM_RC_SCHEME);
}
do
   // Generate a random key pair
if (!BnEccGenerateKeyPair(bnK, ecR, E, rand))
   break;
   // Convert R.x to a string
BnTo2B(ecR->x, e, (NUMBYTES)BITS_TO_BYTES(BnSizeInBits(prime)));

   // f) compute r = Hash(e || P) (mod n)
CryptHashStart(&hashState, hashAlg);
CryptDigestUpdate2B(&hashState, e);
CryptDigestUpdate2B(&hashState, &digest->b);
   e->size = CryptHashEnd(&hashState, digestSize, e->buffer);
   // Reduce the hash size if it is larger than the curve order
SchnorrReduce(e, order);
   // Convert hash to number
BnFrom2B(bnR, e);
   // Do the Schnorr computation
retVal = BnSchnorrSign(bnS, bnK, bnR, bnD, CurveGetOrder(C));
while (retVal == TPM_RC_NO_RESULT);

Exit:
return retVal;

#endif // ALG_ECSCHNORR
#endif _SM2_SIGN_DEBUG

10.2.12.3.5 BnHexEqual()

This function compares a bignum value to a hex string.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>values equal</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>values not equal</td>
</tr>
</tbody>
</table>

static BOOL BnHexEqual(
   bigNum bn,     //IN: big number value
10.2.12.3.6 BnSignEcSm2()

This function signs a digest using the method defined in SM2 Part 2. The method in the standard will add
a header to the message to be signed that is a hash of the values that define the key. This then hashed
with the message to produce a digest (e) that is signed. This function signs e.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>bad curve</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
BnSignEcSm2(
    bigNum                   bnR,   // OUT: r component of the signature
    bigNum                   bnS,   // OUT: s component of the signature
    bigCurve                 E,     // IN: the curve used in signing
    bigNum                   bnD,   // IN: the private key
    const TPM2B_DIGEST      *digest, // IN: the digest to sign
    RAND_STATE              *rand   // IN: random number generator (mostly for
d                                                                  //     debug)
) {
    BN_MAX_INITIALIZED(bnE, digest);   // Don't know how big digest might be
    ECC_NUM(bnN);                       
    ECC_NUM(bnK);                       
    ECC_NUM(bnT);                       // temp
    POINT(Q1);
    bigConst                  order = (E != NULL)
                                ? CurveGetOrder(AccessCurveData(E)) : NULL;
    // A3: Use random number generator to generate random number 1 <= k <= n-1;
    // NOTE: Ax: numbers are from the SM2 standard
    loop:
    {                               
        // Get a random number 0 < k < n
        BnGenerateRandomInRange(bnK, order, rand);
        #ifdef _SM2_SIGN_DEBUG
        BnFromHex(bnE, "B524F552CD82B8B028476E005C377FB1"
                                "9A87E6FC62D48BB5D42E3D9B9EFE7E7");
        BnFromHex(bnD, "128B2FABB0433C6C068CD803DFF7979"
                                "2A519A55171B1B650C23661D15897263");
        #endif
        // A3: Use random number generator to generate random number 1 <= k <= n-1;
        // NOTE: Ax: numbers are from the SM2 standard
        loop:
        {                               
            // Get a random number 0 < k < n
            BnGenerateRandomInRange(bnK, order, rand);
            #ifdef _SM2_SIGN_DEBUG
            BnFromHex(bnE, "B524F552CD82B8B028476E005C377FB1"
                                "9A87E6FC62D48BB5D42E3D9B9EFE7E7");
            BnFromHex(bnD, "128B2FABB0433C6C068CD803DFF7979"
                                "2A519A55171B1B650C23661D15897263");
            #endif
            // A4: Figure out the point of elliptic curve (x1, y1)=[k]G, and according
            // to details specified in 4.2.7 in Part 1 of this document, transform the
            // data type of x1 into an integer;
            if(!BnEccModMult(Q1, NULL, bnK, E))
                goto loop;
            // A5: Figure out r = (e + x1) mod n,
            BnAdd(bnR, bnE, Q1->x);
            BnMod(bnR, order);
            #ifdef _SM2_SIGN_DEBUG
            pAssert(BnHexEqual(bnR, "40F1EC59F793D9F49E09DCEF49130D41"
                                "F1EC59F793D9F49E09DCEF49130D41");
```
#ifdef __GNUC__
// if r=0 or r+k=n, return to A3;
if (BnEqualZero(bnR))
  goto loop;
BnAdd(bnT, bnK, bnR);
if (BnUnsignedCmp(bnT, bnN) == 0)
  goto loop;
// A6: Figure out s = ((1 + dA)^-1 (k - r * dA)) mod n,
// if s=0, return to A3;
// compute t = (1+dA)^-1
BnAddWord(bnT, bnD, 1);
BnModInverse(bnT, bnT, order);

#elif defined _SM2_SIGN_DEBUG
  pAssert(BnHexEqual(bnT, "79BFCF3052C80DA7B939B0C6914A18CB"
               "B2D96D8555256E83122743A7D4F5F956"));
#endif

// compute s = t * (k - r * dA) mod n
BnModMult(bnS, bnR, bnD, order);
// k - r * dA mod n = k + n - ((r * dA) mod n)
BnSub(bnS, order, bnS);
BnAdd(bnS, bnK, bnS);
BnModMult(bnS, bnS, bnT, order);

#elif defined _SM2_SIGN_DEBUG
  pAssert(BnHexEqual(bnS, "6FC6DA32C5D5C5F10C77DB20F7C2EB6"
                    "67A457872FB09EC56327A67EC7DEEBE7"));
#endif

if (BnEqualZero(bnS))
  goto loop;

  } // A7: According to details specified in 4.2.1 in Part 1 of this document,
// transform the data type of r, s into bit strings, signature of message M
// is (r, s).
// This is handled by the common return code

#elif defined _SM2_SIGN_DEBUG
  pAssert(BnHexEqual(bnR, "40F1EC59F793D9F49E09DEEF49130D41"
                     "94F79FB1EED2CAA555BACDDB49C4E755D1"));
  pAssert(BnHexEqual(bnS, "6FC6DA32C5D5C5F10C77DB20F7C2EB6"
                     "67A457872FB09EC56327A67EC7DEEBE7"));
#endif

return TPM_RC_SUCCESS;

#endif // ALG_SM2
TPMT_ECC_SCHEME *scheme, // IN: signing scheme
RAND_STATE *rand
}

CURVE_INITIALIZED(E, signKey->publicArea.parameters.eccDetail.curveID);
ECC_INITIALIZED(bnD, &signKey->sensitive.sensitive.ecc.b);
ECC_NUM(bnR);
ECC_NUM(bnS);
const ECC_CURVE_DATA *C;
TPM_RC retVal = TPM_RC_SCHEME;

// NOT REFERENCED(scheme);
if (E == NULL)
    ERROR_RETURN(TPM_RC_VALUE);
C = AccessCurveData(E);
signature->signature.ecdaa.signatureR.t.size
    = sizeof(signature->signature.ecdaa.signatureR.t.buffer);
signature->signature.ecdaa.signatureS.t.size
    = sizeof(signature->signature.ecdaa.signatureS.t.buffer);
TEST(signature->sigAlg);
switch (signature->sigAlg)
{
case ALG_ECDSA_VALUE:
    retVal = BnSignEcdsa(bnR, bnS, E, bnD, digest, rand);
    break;
#if ALG_ECDAA
    case ALG_ECDAA_VALUE:
        retVal = BnSignEcdaa(&signature->signature.ecdaa.signatureR, bnS, E,
                            bnD, digest, scheme, signKey, rand);
        bnR = NULL;
        break;
#endif
#if ALG_ECSCHNORR
    case ALG_ECSCHNORR_VALUE:
        retVal = BnSignEcSchnorr(bnR, bnS, E, bnD, digest,
                                 signature->signature.ecschnorr.hash, rand);
        break;
#endif
#if ALG_SM2
    case ALG_SM2_VALUE:
        retVal = BnSignEcSm2(bnR, bnS, E, bnD, digest, rand);
        break;
#endif
    default:
        break;
}
// If signature generation worked, convert the results.
if (retVal == TPM_RC_SUCCESS)
{
    NUMBYTES orderBytes =
        (NUMBYTES)BITS_TO_BYTES(BnSizeInBits(CurveGetOrder(C)));
    if (bnR != NULL)
        BnTo2B(bnR, &signature->signature.ecdaa.signatureR.b, orderBytes);
    if (bnS != NULL)
        BnTo2B(bnS, &signature->signature.ecdaa.signatureS.b, orderBytes);
    Exit:
    CURVE_FREE(E);
    return retVal;
#if ALG_ECDSA
10.2.12.3.8 \texttt{BnValidateSignatureEcdsa()}

This function validates an ECDSA signature. \textit{rIn} and \textit{sIn} should have been checked to make sure that they are in the range $0 < v < n$.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIGNATURE</td>
<td>signature not valid</td>
</tr>
</tbody>
</table>

```c
TPM\_RC
BnValidateSignatureEcdsa(
bigNum \hspace{1em} \text{bnR}, \hspace{1em} // IN: r component of the signature
bigNum \hspace{1em} \text{bnS}, \hspace{1em} // IN: s component of the signature
curve \hspace{1em} \text{E}, \hspace{1em} // IN: the curve used in the signature
bn\_point\_t \hspace{1em} *ecQ, \hspace{1em} // IN: the public point of the key
const \text{TPM2B\_DIGEST} \hspace{1em} *digest \hspace{1em} // IN: the digest that was signed
)
{
// Make sure that the allocation for the digest is big enough for a maximum
// digest
BN\_VAR(bnE, \text{MAX(MAX\_ECC\_KEY\_BYTES, MAX\_DIGEST\_SIZE}) \times 8);
POINT(ecR);
ECC\_NUM(bnU1);
ECC\_NUM(bnU2);
ECC\_NUM(bnW);
bignum order = CurveGetOrder(AccessCurveData(E));
TPM\_RC
retVal = TPM\_RC\_SIGNATURE;

// Get adjusted digest
EcdsaDigest(bnE, digest, order);
// 1. If r and s are not both integers in the interval [1, n - 1], output
// INVALID.
// bnr and bns were validated by the caller
// 2. Use the selected hash function to compute H0 = Hash(M0).
// This is an input parameter
// 3. Convert the bit string H0 to an integer e as described in Appendix B.2.
// Done at entry
// 4. Compute w = (s')^-1 mod n, using the routine in Appendix B.1.
if(!BnModInverse(bnW, bnS, order))
goto Exit;
// 5. Compute u1 = (e' * w) mod n, and compute u2 = (r' * w) mod n.
BnModMult(bnU1, bnE, bnW, order);
BnModMult(bnU2, bnR, bnW, order);
// 6. Compute the elliptic curve point R = (xR, yR) = u1G+u2Q, using EC
// scalar multiplication and EC addition (see [Routines]). If R is equal to
// the point at infinity O, output INVALID.
if(BnPointMult(ecR, CurveGetG(AccessCurveData(E)), bnU1, ecQ, bnU2, E)
!= TPM\_RC\_SUCCESS)
goto Exit;
// 7. Compute v = Rx mod n.
BnMod(ecR->x, order);
// 8. Compare v and r0. If v = r0, output VALID; otherwise, output INVALID
if(BnUnsignedCmp(ecR->x, bnR) != 0)
goto Exit;
retVal = TPM\_RC\_SUCCESS;
Exit:
return retVal;
}
```

```c
#endif // ALG\_ECDSA
```

```c
#if ALG\_SM2
```
10.2.12.3.9  BnValidateSignatureEcSm2()

This function is used to validate an SM2 signature.

Error Returns | Meaning
---|---
TPM_RC_SIGNATURE | signature not valid

```c
static TPM_RC
static TPM_RC BnValidateSignatureEcSm2(
  bigNum      bnR,      // IN: r component of the signature
  bigNum      bnS,      // IN: s component of the signature
  bigCurve    E,        // IN: the curve used in the signature process
  bigPoint    ecQ,      // IN: the public point of the key
  const TPM2B_DIGEST *digest  // IN: the digest that was signed
){
  POINT(P);
  ECC_NUM(bnRp);
  ECC_NUM(bnT);
  BN_MAX_INITIALIZED(bnE, digest);
  BOOL _OK;
  bigConst
    order = CurveGetOrder(AccessCurveData(E));
  #ifdef _SM2_SIGN_DEBUG
    // Make sure that the input signature is the test signature
    pAssert(BnHexEqual(bnR, "40F1EC59F793D9F49E09DCEF49130D41"
                   "94F79FB1EED2CA55BACDB49C4E755D1");
    pAssert(BnHexEqual(bnS, "6FC6DAC32C5D5CF10C77DFB20F7C2EB6"
                   "67A457872FB09EC56327A67EC7DEEBE7");
  #endif
  // b) compute t := (r + s) mod n
  BnAdd(bnT, bnR, bnS);
  BnMod(bnT, order);
  #ifdef _SM2_SIGN_DEBUG
    pAssert(BnHexEqual(bnT, "2B75F07E7ECE7CC1C8986B991F441A"
                   "D324D6619FE06DD63ED32EC997C801");
  #endif
  // c) verify that t > 0
  OK = !BnEqualZero(bnT);
  if(!OK)
    // set T to a value that should allow rest of the computations to run
    // without trouble
    BnCopy(bnT, bnS);
  // d) compute (x, y) := [s]G + [t]Q
  OK = BnEccModMult2(P, NULL, bnS, ecQ, bnT, E);
  #ifdef _SM2_MOD2_DEBUG
    pAssert(OK && BnHexEqual(P->x, "110FCDA57615705D5E7B9324AC4B856D"
                      "23E6D9188B2AE47759514657CE25D112");
  #endif
  // e) compute r' := (e + x) mod n (the x coordinate is in bnT)
  OK = OK && BnAdd(bnRp, bnE, P->x);
  OK = OK && BnMod(bnRp, order);
  // f) verify that r' = r
  OK = OK && (BnUnsignedCmp(bnR, bnRp) == 0);
  if(!OK)
    return TPM_RC_SIGNATURE;
else
```

---
return TPM_RC_SUCCESS;
}
#endif  // ALG_SM2
#endif  // ALG_ECSCHNORR

10.2.12.3.10 BnValidateSignatureEcSchnorr()

This function is used to validate an EC Schnorr signature.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIGNATURE</td>
<td>signature not valid</td>
</tr>
</tbody>
</table>

static TPM_RC
BnValidateSignatureEcSchnorr(
bigNum               bnR,  // IN: r component of the signature
bigNum               bnS,  // IN: s component of the signature
TPM_ALG_ID           hashAlg, // IN: hash algorithm of the signature
bigCurve             E,    // IN: the curve used in the signature
bigPoint             ecQ,   // IN: the public point of the key
const TPM2B_DIGEST  *digest // IN: the digest that was signed
)
{
    BN_MAX(bnRn);
    POINT(ecE);
    BN_MAX(bnEx);
    const ECC_CURVE_DATA    *C = AccessCurveData(E);
    bigConst                 order = CurveGetOrder(C);
    UINT16                   digestSize = CryptHashGetDigestSize(hashAlg);
    HASH_STATE               hashState;
    TPM2B_TYPE(BUFFER, MAX(MAX_ECC_PARAMETER_BYTES, MAX_DIGEST_SIZE));
    TPM2B_BUFFER             Ex2 = {{sizeof(Ex2.t.buffer),0}};
    BOOL                     OK = BnPointMult(ecE, CurveGetG(C), bnS, ecQ, bnRn, E) == TPM_RC_SUCCESS;
    // reduce the x portion of E mod q
    OK = OK && BnMod(ecE->x, order);
    // Convert to byte string
    OK = OK && BnTo2B(ecE->x, &Ex2.b,
                      (NUMBYTES)(BITS_TO_BYTES(BnSizeInBits(order))));
    if(OK)
    {
        // Ex = h(pE.x || digest)
        CryptHashStart(&hashState, hashAlg);
        CryptDigestUpdate(&hashState, Ex2.t.size, Ex2.t.buffer);
        CryptDigestUpdate(&hashState, digest->t.size, digest->t.buffer);
        Ex2.t.size = CryptHashEnd(&hashState, digestSize, Ex2.t.buffer);
        SchnorrReduce(&Ex2.b, order);
        BnFrom2B(bnEx, &Ex2.b);
        // see if Ex matches R
        OK = BnUnsignedCmp(bnEx, bnR) == 0;
    }
    return (OK) ? TPM_RC_SUCCESS : TPM_RC_SIGNATURE;
}  // ALG_ECSCHNORR
### 10.2.12.3.11 CryptEccValidateSignature()

This function validates an EcDsa() or EcSchnorr() signature. The point Qin needs to have been validated to be on the curve of curveId.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIGNATURE</td>
<td>not a valid signature</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT TPM_RC
CryptEccValidateSignature(
    // IN: signature to be verified
    TPMT_SIGNATURE *signature,
    // IN: ECC key signed the hash
    OBJECT *signKey,
    // IN: digest that was signed
    const TPM2B_DIGEST *digest,
)
{
    CURVE_INITIALIZED(E, signKey->publicArea.parameters.eccDetail.curveID);
    ECC_NUM(bnR);
    ECC_NUM(bnS);
    POINT_INITIALIZED(ecQ, &signKey->publicArea.unique.ecc);
    bigConst order;
    TPM_RC retVal;

    if (E == NULL)
        ERROR_RETURN(TPM_RC_VALUE);

    order = CurveGetOrder(AccessCurveData(E));

    switch (signature->sigAlg)
    {
        case ALG_ECDSA_VALUE:
            if (BnEqualZero(bnR) || BnEqualZero(bnS))
                ERROR_RETURN(TPM_RC_SIGNATURE);
            if (BnUnsignedCmp(bnS, order) >= 0)
                ERROR_RETURN(TPM_RC_SIGNATURE);
            break;
        default:break;
    }

    switch (signature->sigAlg)
    {
        case ALG_ECDSA_VALUE:
            retVal = BnValidateSignatureEcdsa(bnR, bnS, E, ecQ, digest);
            break;
        if (ALG_ECSCHNORR)
            case ALG_ECSCHNORR_VALUE:
                retVal = BnValidateSignatureEcSchnorr(bnR, bnS, E, ecQ, digest);
                break;
        #if ALG_SM2
            case ALG_SM2_VALUE:
                retVal = BnValidateSignatureSm2(bnR, bnS, E, ecQ, digest);
                break;
        #endif
    }
    return retVal;
}
```
10.2.12.3.12 CryptEccCommitCompute()

This function performs the point multiply operations required by TPM2_Commit().

If B or M is provided, they must be on the curve defined by curveId. This routine does not check that they are on the curve and results are unpredictable if they are not.

It is a fatal error if r is NULL. If B is not NULL, then it is a fatal error if d is NULL or if K and L are both NULL. If M is not NULL, then it is a fatal error if E is NULL.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>if K, L or E was computed to be the point at infinity</td>
</tr>
<tr>
<td>TPM_RC_CANCELED</td>
<td>a cancel indication was asserted during this function</td>
</tr>
</tbody>
</table>

```
LIB_EXPORT TPM_RC CryptEccCommitCompute(
    TPMS_ECC_POINT *K, // OUT: [d]B or [r]Q
    TPMS_ECC_POINT *L, // OUT: [r]B
    TPMS_ECC_POINT *E, // OUT: [r]M
    TPM_ECC_CURVE curveId, // IN: the curve for the computations
    TPMS_ECC_POINT *M, // IN: M (optional)
    TPMS_ECC_POINT *B, // IN: B (optional)
    TPM2B_ECC_PARAMETER *d, // IN: d (optional)
    TPM2B_ECC_PARAMETER *r // IN: the computed r value (required)
)
{
    CURVE_INITIALIZED(curveId); // Normally initialize E as the curve, but
    ECC_INITIALIZED(bnR, r); // E means something else in this function
    TPM_RC retval = TPM_RC_SUCCESS;
    // Validate that the required parameters are provided.
    pAssert(r != NULL && E != NULL);
    // Note: E has to be provided if computing E := [r]Q or E := [r]M. Will do
    // E := [r]Q if both M and B are NULL.
    pAssert(r != NULL && E != NULL);
    // Initialize the output points in case they are not computed
    ClearPoint2B(K);
    ClearPoint2B(L);
    ClearPoint2B(E);
    // Sizes of the r parameter may not be zero
    pAssert(r->t.size > 0);
    // If B is provided, compute K=[d]B and L=[r]B
    if (B != NULL)
```
{  
    ECC_INITIALIZED(bnD, d);
    POINT_INITIALIZED(pB, B);
    POINT(pK);
    POINT(pL);

    // pAssert(d != NULL && K != NULL && L != NULL);

    if(!BnIsOnCurve(pB, AccessCurveData(curve)))
        ERROR_RETURN(TPM_RC_VALUE);
    // do the math for \( K = [d]B \)
    if((retVal = BnPointMult(pK, pB, bnD, NULL, NULL, null)) != TPM_RC_SUCCESS)
        goto Exit;
    // Convert BN K to TPM2B K
    BnPointTo2B(K, pK, curve);
    // do the math for \( L = [r]B \)
    if(!_plat__IsCanceled())
    {
        BnIsValidPrivateEcc(bnR, curve);
        if((retVal = BnPointMult(pL, pB, bnR, NULL, NULL, curve)) != TPM_RC_SUCCESS)
            goto Exit;
        BnPointTo2B(L, pL, curve);
    }
    // Convert BN L to TPM2B L
    BnPointTo2B(L, pL, curve);

    if((M != NULL) || (B == NULL))
    {
        POINT_INITIALIZED(pM, M);
        POINT(pE);

        // Make sure that a place was provided for the result
        pAssert(E != NULL);

        // if this is the third point multiply, check for cancel first
        if((B != NULL) && _plat__IsCanceled())
            ERROR_RETURN(TPM_RC_CANCELED);
    }
    // If M provided, then pM will not be NULL and will compute \( E = [r]M \).
    // However, if M was not provided, then pM will be NULL and \( E = [r]G \)
    // will be computed
    if((retVal = BnPointMult(pE, pM, bnR, NULL, NULL, curve)) != TPM_RC_SUCCESS)
        goto Exit;
    // Convert E to 2B format
    BnPointTo2B(E, pE, curve);

    Exit:
    CURVE_FREE(curve);
    return retVal;
}
#endif // ALG_ECC
10.2.13 CryptHash.c

10.2.13.1 Description

This file contains implementation of cryptographic functions for hashing.

10.2.13.2 Includes, Defines, and Types

```c
#define _CRYPT_HASH_C_
#include "TpSetup.h"
define HASH_TABLE_SIZE (HASH_COUNT + 1)
extern const HASH_INFO g_hashData[HASH_COUNT + 1];
#if ALG_SHA1
HASH_DEF_TEMPLATE(SHA1);
#endif
#if ALG_SHA256
HASH_DEF_TEMPLATE(SHA256);
#endif
#if ALG_SHA384
HASH_DEF_TEMPLATE(SHA384);
#endif
#if ALG_SHA512
HASH_DEF_TEMPLATE(SHA512);
#endif
HASH_DEF nullDef = {{0}};
```

10.2.13.3 Obligatory Initialization Functions

10.2.13.3.1 CryptHashInit()

This function is called by _TPM_Init() do perform the initialization operations for the library.

```c
BOOL CryptHashInit()
void
{
 LibHashInit();
 return TRUE;
}
```

10.2.13.3.2 CryptHashStartup()

This function is called by TPM2_Startup() in case there is work to do at startup. Currently, this is a placeholder.

```c
BOOL CryptHashStartup()
void
{
 return TRUE;
}
```
10.2.13.4 Hash Information Access Functions

10.2.13.4.1 Introduction

These functions provide access to the hash algorithm description information.

10.2.13.4.2 CryptGetHashDef()

This function accesses the hash descriptor associated with a hash algorithm. The function returns NULL for TPM_ALG_NULL and fails if hashAlg is not a hash algorithm.

```c
PHASH_DEF
CryptGetHashDef(
    TPM_ALG_ID       hashAlg
)
{
    PHASH_DEF       retVal;
    switch(hashAlg)
    {
        #if ALG_SHA1
            case ALG_SHA1_VALUE:
                return &SHA1_Def;
                break;
        #endif
        #if ALG_SHA256
            case ALG_SHA256_VALUE:
                retVal = &SHA256_Def;
                break;
        #endif
        #if ALG_SHA384
            case ALG_SHA384_VALUE:
                retVal = &SHA384_Def;
                break;
        #endif
        #if ALG_SHA512
            case ALG_SHA512_VALUE:
                retVal = &SHA512_Def;
                break;
        #endif
        default:
            retVal = &nullDef;
            break;
    }
    return retVal;
}
```

10.2.13.4.3 CryptHashIsValidAlg()

This function tests to see if an algorithm ID is a valid hash algorithm. If flag is true, then TPM_ALG_NULL is a valid hash.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>hashAlg is a valid, implemented hash on this TPM</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>hashAlg is not valid for this TPM</td>
</tr>
</tbody>
</table>

```c
BOOL
CryptHashIsValidAlg(
    TPM_ALG_ID       hashAlg,
    BOOL             flag
)
```
71 }  
72 {  
73 switch(hashAlg)  
74 {  
75 #if ALG_SHA1  
76 case ALG_SHA1_VALUE:  
77 #endif  
78 #if ALG_SHA256  
79 case ALG_SHA256_VALUE:  
80 #endif  
81 #if ALG_SHA384  
82 case ALG_SHA384_VALUE:  
83 #endif  
84 #if ALG_SHA512  
85 case ALG_SHA512_VALUE:  
86 #endif  
87 #if ALG_SM3_256  
88 case ALG_SHA256_VALUE:  
89 #endif  
90 return TRUE;  
91 break;  
92 case ALG_NULL_VALUE:  
93 return flag;  
94 break;  
95 default:  
96 break;  
97 }  
98 return FALSE;  
99 }

10.2.13.4.4 GetHashInfoPointer()

This function returns a pointer to the hash info for the algorithm. If the algorithm is not supported, function returns a pointer to the data block associated with TPM_ALG_NULL.

NOTE: The data structure must have a digest size of 0 for TPM_ALG_NULL.

static const HASH_INFO * GetHashInfoPointer(  
    TPM_ALG_ID hashAlg  
)  
{  
    UINT32 i;  
    // ALG_NULL is the stop value so search up to it  
    for(i = 0; i < HASH_COUNT; i++)  
    {  
        if(g_hashData[i].alg == hashAlg)  
            return &g_hashData[i];  
    }  
    // either the input was TPM_ALG_NULL or we didn't find the requested algorithm  
    // in either case return a pointer to the TPM_ALG_NULL "hash" descriptor  
    return &g_hashData[HASH_COUNT];  
}

10.2.13.4.5 CryptHashGetAlgByIndex()

This function is used to iterate through the hashes. TPM_ALG_NULL is returned for all indexes that are not valid hashes. If the TPM implements 3 hashes, then an index value of 0 will return the first implemented hash and an index of 2 will return the last. All other index values will return TPM_ALG_NULL.
## 10.2.13.4.6 CryptHashGetDigestSize()

Returns the size of the digest produced by the hash. If `hashAlg` is not a hash algorithm, the TPM will FAIL.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TPM_ALG_NULL</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>the digest size</td>
</tr>
</tbody>
</table>

```c
LIBEXPORT UINT16 CryptHashGetDigestSize(
    TPM_ALG_ID hashAlg,  // IN: hash algorithm to look up
)
{
    return CryptGetHashDef(hashAlg)->digestSize;
}
```

## 10.2.13.4.7 CryptHashGetBlockSize()

Returns the size of the block used by the hash. If `hashAlg` is not a hash algorithm, the TPM will FAIL.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TPM_ALG_NULL</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>the digest size</td>
</tr>
</tbody>
</table>

```c
LIBEXPORT UINT16 CryptHashGetBlockSize(
    TPM_ALG_ID hashAlg,  // IN: hash algorithm to look up
)
{
    return CryptGetHashDef(hashAlg)->blockSize;
}
```

## 10.2.13.4.8 CryptHashGetDer

This function returns a pointer to the DER string for the algorithm and indicates its size.

```c
LIBEXPORT UINT16 CryptHashGetDer(
    TPM_ALG_ID hashAlg,  // IN: the algorithm to look up
    const BYTE **p
)
145 }  
146 {  
147   const HASH_INFO *q;  
148   q = GetHashInfoPointer(hashAlg);  
149   *p = &q->der[0];  
150   return q->derSize;  
151 }

10.2.13.4.9 CryptHashGetContextAlg()

This function returns the hash algorithm associated with a hash context.

152 TPM_ALG_ID  
153 CryptHashGetContextAlg(  
154   PHASH_STATE state // IN: the context to check  
155 }  
156 {  
157   return state->hashAlg;  
158 }

10.2.13.5 State Import and Export

10.2.13.5.1 CryptHashCopyState

This function is used to clone a HASH_STATE.

159 LIB_EXPORT void  
160 CryptHashCopyState(  
161   HASH_STATE *out, // OUT: destination of the state  
162   const HASH_STATE *in // IN: source of the state  
163 }  
164 {  
165   pAssert(out->type == in->type);  
166   out->hashAlg = in->hashAlg;  
167   out->def = in->def;  
168   if(in->hashAlg != TPM_ALG_NULL)  
169   {  
170     // Just verify that the hashAlg makes sense (is implemented)  
171     CryptGetHashDef(in->hashAlg);  
172     // ... and copy.  
173     HASH_STATE_COPY(out, in);  
174   }  
175   if(in->type == HASH_STATE_HMAC)  
176   {  
177     const HMAC_STATE *hIn = (HMAC_STATE *)in;  
178     HMAC_STATE *hOut = (HMAC_STATE *)out;  
179     hOut->hmacKey = hIn->hmacKey;  
180   }  
181   return;  
182 }

10.2.13.5.2 CryptHashExportState()

This function is used to export a hash or HMAC hash state. This function would be called when preparing to context save a sequence object.

183 void  
184 CryptHashExportState(  
185   PCHASH_STATE internalFmt, // IN: the hash state formatted for use by  
186   // library
PEXPORT_HASH_STATE externalFmt // OUT: the exported hash state

BYTE *outBuf = (BYTE *)externalFmt;

// the following #define is used to move data from an aligned internal data
// structure to a byte buffer (external format data.
#define CopyToOffset(value) \
    memcpy(&outBuf[offsetof(HASH_STATE, value)], &internalFmt->value, \
    sizeof(internalFmt->value))

// Copy the hashAlg
CopyToOffset(hashAlg);
CopyToOffset(type);

#ifdef HASH_STATE_SMAC
    if (internalFmt->type == HASH_STATE_SMAC)
    {
        memcpy(outBuf, internalFmt, sizeof(HASH_STATE));
        return;
    }
#endif

if (internalFmt->type == HASH_STATE_HMAC)
{
    HMAC_STATE *from = (HMAC_STATE *)internalFmt;
    memcpy(&outBuf[offsetof(HMAC_STATE, hmacKey)], &from->hmacKey, 
    sizeof(from->hmacKey));
    if (internalFmt->hashAlg != TPM_ALG_NULL)
    { 
        HASH_STATE_EXPORT(externalFmt, internalFmt);
    }
}

10.2.13.5.3 CryptHashImportState()

This function is used to import the hash state. This function would be called to import a hash state when the context of a sequence object was being loaded.

void CryptHashImportState( 
    PHASH_STATE internalFmt, // OUT: the hash state formatted for use by 
    // the library 
    PCEXPORT_HASH_STATE externalFmt // IN: the exported hash state 
)
{
    BYTE *inBuf = (BYTE *)externalFmt;

    // Copy the hashAlg of the byte-aligned input structure to the structure-aligned 
    // internal structure.
    CopyFromOffset(hashAlg);
    CopyFromOffset(type);
    if (internalFmt->hashAlg != TPM_ALG_NULL)
    {
        #ifdef HASH_STATE_SMAC
            if (internalFmt->type == HASH_STATE_SMAC)
            {
                memcpy(internalFmt, inBuf, sizeof(HASH_STATE));
                return;
            }
        #endif

        internalFmt->def = CryptGetHashDef(internalFmt->hashAlg);
10.2.13.6 State Modification Functions

10.2.13.6.1 HashEnd()

Local function to complete a hash that uses the hashDef instead of an algorithm ID. This function is used to complete the hash and only return a partial digest. The return value is the size of the data copied.

```c
static UINT16
HashEnd(  
    PHASH_STATE      hashState,  // IN: the hash state
    UINT32           dOutSize,  // IN: the size of receive buffer
    PBYTE            dOut,     // OUT: the receive buffer
)
{
    BYTE                temp[MAX_DIGEST_SIZE];

    if( (hashState->hashAlg == TPM_ALG_NULL)  
        || (hashState->type != HASH_STATE_HASH) )
    {
        dOutSize = 0;

        if(dOutSize > 0)
        {
            hashState->def = CryptGetHashDef(hashState->hashAlg);
            // Set the final size
            dOutSize = MIN(dOutSize, hashState->def->digestSize);
            // Complete into the temp buffer and then copy
            HASH_END(hashState, temp);
            // Don't want any other functions calling the HASH_END method
            // directly.
            #undef HASH_END
            memcpy(dOut, &temp, dOutSize);
        }
    }

    hashState->type = HASH_STATE_EMPTY;
    return (UINT16)dOutSize;
}
```

10.2.13.6.2 CryptHashStart()

Functions starts a hash stack Start a hash stack and returns the digest size. As a side effect, the value of stateSize in hashState is updated to indicate the number of bytes of state that were saved. This function calls GetHashServer() and that function will put the TPM into failure mode if the hash algorithm is not supported.

This function does not use the sequence parameter. If it is necessary to import or export context, this will start the sequence in a local state and export the state to the input buffer. Will need to add a flag to the state structure to indicate that it needs to be imported before it can be used. (BLEH).
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hash is TPM_ALG_NULL</td>
</tr>
<tr>
<td>&gt;0</td>
<td>digest size</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT UINT16
CryptHashStart(
    PHASH_STATE    hashState,       // OUT: the running hash state
    TPM_ALG_ID     hashAlg         // IN: hash algorithm
)
{
    UINT16        retVal;
    TEST(hashAlg);
    hashState->hashAlg = hashAlg;
    if(hashAlg == TPM_ALG_NULL)
    {
        retVal = 0;
    }
    else
    {
        hashState->def = CryptGetHashDef(hashAlg);
        HASH_START(hashState);
        retVal = hashState->def->digestSize;
    }
    #undef HASH_START
    hashState->type = HASH_STATE_HASH;
    return retVal;
}
```

**10.2.13.6.3 CryptDigestUpdate()**

Add data to a hash or HMAC, SMAC stack.

```c
void CryptDigestUpdate(
    PHASH_STATE    hashState,       // IN: the hash context information
    UINT32         dataSize,        // IN: the size of data to be added
    const BYTE    *data              // IN: data to be hashed
)
{
    if(hashState->hashAlg != TPM_ALG_NULL)
    {
        if((hashState->type == HASH_STATE_HASH)
            || (hashState->type == HASH_STATE_HMAC))
            HASH_DATA(hashState, dataSize, (BYTE *)data);
        #if SMAC_IMPLEMENTED
        else if(hashState->type == HASH_STATE_SMAC)
            (hashState->state.smac.smacMethods.data)(&hashState->state.smac.state,
                dataSize, data);
        #endif // SMAC_IMPLEMENTED
        else
            FAIL(FATAL_ERROR_INTERNAL);
    }
    return;
}
```
10.2.13.6.4 CryptHashEnd()

Complete a hash or HMAC computation. This function will place the smaller of digestSize or the size of the digest in dOut. The number of bytes in the placed in the buffer is returned. If there is a failure, the returned value is <= 0.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>no data returned</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>the number of bytes in the digest or dOutSize, whichever is smaller</td>
</tr>
</tbody>
</table>

LIB_EXPORT UINT16 CryptHashEnd(
PHASH_STATE hashState, // IN: the state of hash stack
UINT32 dOutSize, // IN: size of digest buffer
BYTE *dOut // OUT: hash digest
)

10.2.13.6.5 CryptHashBlock()

Start a hash, hash a single block, update digest and return the size of the results.

The digestSize parameter can be smaller than the digest. If so, only the more significant bytes are returned.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= 0</td>
<td>number of bytes placed in dOut</td>
</tr>
</tbody>
</table>

LIB_EXPORT UINT16 CryptHashBlock(
TPM_ALG_ID hashAlg, // IN: The hash algorithm
UINT32 dataSize, // IN: size of buffer to hash
const BYTE *data, // IN: the buffer to hash
UINT32 dOutSize, // IN: size of the digest buffer
BYTE *dOut // OUT: digest buffer
)

10.2.13.6.6 CryptDigestUpdate2B()

This function updates a digest (hash or HMAC) with a TPM2B.

This function can be used for both HMAC and hash functions so the digestState is void so that either state type can be passed.

LIB_EXPORT void CryptDigestUpdate2B(
PHASH_STATE state, // IN: the digest state
const TPM2B *bIn // IN: 2B containing the data
)

Only compute the digest if a pointer to the 2B is provided.

In CryptDigestUpdate(), if size is zero or buffer is NULL, then no change to the digest occurs. This function should not provide a buffer if bIn is not provided.
pAssert(bIn != NULL);
CryptDigestUpdate(state, bIn->size, bIn->buffer);
return;
}

10.2.13.6.7 CryptHashEnd2B()

This function is the same as CryptCompleteHash() but the digest is placed in a TPM2B. This is the most common use and this is provided for specification clarity. ‘digest.size’ should be set to indicate the number of bytes to place in the buffer

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=0</td>
<td>the number of bytes placed in 'digest.buffer'</td>
</tr>
</tbody>
</table>

```
LIB_EXPORT UINT16
CryptHashEnd2B(
    PHASH_STATE      state, // IN: the hash state
    P2B              digest // IN: the size of the buffer Out: requested
    // number of bytes
)
{
    return CryptHashEnd(state, digest->size, digest->buffer);
}
```

10.2.13.6.8 CryptDigestUpdateInt()

This function is used to include an integer value to a hash stack. The function marshals the integer into its canonical form before calling CryptDigestUpdate().

```
LIB_EXPORT void
CryptDigestUpdateInt(
    void     *state,     // IN: the state of hash stack
    UINT32          intSize, // IN: the size of 'intValue' in bytes
    UINT64          intValue // IN: integer value to be hashed
)
{
    #if LITTLE_ENDIAN_TPM
    intValue = REVERSE_ENDIAN_64(intValue);
    #endif
    CryptDigestUpdate(state, intSize, &((BYTE *)&intValue)[8 - intSize]);
}
```

10.2.13.7 HMAC Functions

10.2.13.7.1 CryptHmacStart

This function is used to start an HMAC using a temp hash context. The function does the initialization of the hash with the HMAC key XOR iPad and updates the HMAC key XOR oPad.

The function returns the number of bytes in a digest produced by hashAlg.
## Return Value

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= 0</td>
<td>number of bytes in digest produced by $hashAlg$ (may be zero)</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT UINT16
CryptHmacStart(
    PHMAC_STATE state,  // IN/OUT: the state buffer
    TPM_ALG_ID hashAlg,  // IN: the algorithm to use
    UINT16 keySize,  // IN: the size of the HMAC key
    const BYTE *key    // IN: the HMAC key
)

// hashDef = CryptGetHashDef(hashAlg);
if(hashDef->digestSize != 0)
{
    // If the HMAC key is larger than the hash block size, it has to be reduced
    // to fit. The reduction is a digest of the hmacKey.
    if(keySize > hashDef->blockSize)
    {
        // if the key is too big, reduce it to a digest of itself
        state->hmacKey.t.size = CryptHashBlock(hashAlg, keySize, key,
                                              hashDef->digestSize,
                                              state->hmacKey.t.buffer);
    }
    else
    {
        memcpy(state->hmacKey.t.buffer, key, keySize);
    }

    // XOR the key with iPad (0x36)
    pb = state->hmacKey.t.buffer;
    for(i = state->hmacKey.t.size; i > 0; i--)
    *

    // if the keySize is smaller than a block, fill the rest with 0x36
    for(i = hashDef->blockSize - state->hmacKey.t.size; i > 0; i--)
    *

    // Increase the oPadSize to a full block
    state->hmacKey.t.size = hashDef->blockSize;

    // Start a new hash with the HMAC key
    // This will go in the caller’s state structure and may be a sequence or not
    CryptHashStart((PHASH_STATE)state, hashAlg);
    CryptDigestUpdate((PHASH_STATE)state, state->hmacKey.t.size,
                      state->hmacKey.t.buffer);

    // XOR the key block with 0x5c ^ 0x36
    for(pb = state->hmacKey.t.buffer, i = hashDef->blockSize; i > 0; i--)
    *

    // Set the hash algorithm
    state->hashState.hashAlg = hashAlg;
    // Set the hash state type
    state->hashState.type = HASH_STATE_HMAC;
    return hashDef->digestSize;
}
10.2.13.7.2 CryptHmacEnd()

This function is called to complete an HMAC. It will finish the current digest, and start a new digest. It will then add the oPadKey and the completed digest and return the results in dOut. It will not return more than dOutSize bytes.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;= 0</td>
<td>number of bytes in dOut (may be zero)</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT UINT16
CryptHmacEnd(
    PHMAC_STATE state, // IN: the hash state buffer
    UINT32 dOutSize, // IN: size of digest buffer
    BYTE *dOut // OUT: hash digest
)
```

10.2.13.7.3 CryptHmacStart2B()

This function starts an HMAC and returns the size of the digest that will be produced.

This function is provided to support the most common use of starting an HMAC with a TPM2B key.

The caller must provide a block of memory in which the hash sequence state is kept. The caller should not alter the contents of this buffer until the hash sequence is completed or abandoned.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 0</td>
<td>the digest size of the algorithm</td>
</tr>
<tr>
<td>= 0</td>
<td>the hashAlg was TPM_ALG_NULL</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT UINT16
CryptHmacStart2B(
```
481 PHMAC_STATE hmacState, // OUT: the state of HMAC stack. It will be used
482 TPMI_ALG_HASH hashAlg, // IN: hash algorithm
483 P2B key // IN: HMAC key
484 }
485 { return CryptHmacStart(hmacState, hashAlg, key->size, key->buffer);
486 }

10.2.13.7.4 CryptHmacEnd2B()

This function is the same as CryptHmacEnd() but the HMAC result is returned in a TPM2B which is the
most common use.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=0</td>
<td>the number of bytes placed in digest</td>
</tr>
</tbody>
</table>

LIB_EXPORT UINT16

CryptHmacEnd2B(
490 491 PHMAC_STATE hmacState, // IN: the state of HMAC stack
492 P2B digest // OUT: HMAC
493 }
494 { return CryptHmacEnd(hmacState, digest->size, digest->buffer);
495 }

10.2.13.8 Mask and Key Generation Functions

10.2.13.8.1 CryptMGF1()

This function performs MGF1 using the selected hash. MGF1 is T(n) = T(n-1) || H(seed || counter). This
function returns the length of the mask produced which could be zero if the digest algorithm is not supported.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hash algorithm was TPM_ALG_NULL</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>should be the same as mSize</td>
</tr>
</tbody>
</table>

LIB_EXPORT UINT16

CryptMGF1(
498 499 UINT32 mSize, // IN: length of the mask to be produced
500 BYTE *mask, // OUT: buffer to receive the mask
501 TPM_ALG_ID hashAlg, // IN: hash to use
502 UINT32 seedSize, // IN: size of the seed
503 BYTE *seed // IN: seed size
504 }
505 { HASH_STATE hashState;
506 PHASH_DEF hDef = CryptGetHashDef(hashAlg);
508 UINT32 remaining;
509 UINT32 counter = 0;
510 BYTE swappedCounter[4];
511 // If there is no digest to compute return
512 if((hashAlg == TPM_ALG_NULL) || (mSize == 0))
513 return 0;
515 for(remaining = mSize; ; remaining -= hDef->digestSize)
517 {
518     // Because the system may be either Endian...
519     UINT32_TO_BYTE_ARRAY(counter, swappedCounter);
520
521     // Start the hash and include the seed and counter
522     CryptHashStart(&hashState, hash Alg);
523     CryptDigestUpdate(&hashState, seedSize, seed);
524     CryptDigestUpdate(&hashState, 4, swappedCounter);
525
526     // Handling the completion depends on how much space remains in the mask
527     // buffer. If it can hold the entire digest, put it there. If not
528     // put the digest in a temp buffer and only copy the amount that
529     // will fit into the mask buffer.
530     HashEnd(&hashState, remaining, mask);
531     if(remaining <= hDef->digestSize)
532         break;
533     mask = &mask[hDef->digestSize];
534     counter++;
535 }
536 return (UINT16)mSize;
537 }

10.2.13.8.2 CryptKDFa()

This function performs the key generation according to Part 1 of the TPM specification.
This function returns the number of bytes generated which may be zero.

The key and keyStream pointers are not allowed to be NULL. The other pointer values may be NULL.
The value of sizeInBits must be no larger than \(2^{18} - 1 = 256K\) bits (32385 bytes).

The once parameter is set to allow incremental generation of a large value. If this flag is TRUE, sizeInBits will be used in the HMAC computation but only one iteration of the KDF is performed. This would be used for XOR obfuscation so that the mask value can be generated in digest-sized chunks rather than having to be generated all at once in an arbitrarily large buffer and then XORed() into the result. If once is TRUE, then sizeInBits must be a multiple of 8.

Any error in the processing of this command is considered fatal.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hash algorithm is not supported or is TPM_ALG_NULL</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>the number of bytes in the keyStream buffer</td>
</tr>
</tbody>
</table>

LIB_EXPORT UINT16
CryptKDFa(  
TPM_ALG_ID hash Alg,  // IN: hash algorithm used in HMAC
const TPM2B *key,  // IN: HMAC key
const TPM2B *label,  // IN: a label for the KDF
const TPM2B *contextU,  // IN: context U
const TPM2B *contextV,  // IN: context V
UINT32 sizeInBits,  // IN: size of generated key in bits
BYTE *keyStream,  // OUT: key buffer
UINT32 *counterInOut,  // IN/OUT: caller may provide the iteration
// counter for incremental operations to
// avoid large intermediate buffers.
UINT16 blocks  // IN: If non-zero, this is the maximum number
// of blocks to be returned, regardless
// of sizeInBits
)
{
    UINT32 counter = 0;  // counter value
    INT16 bytes;  // number of bytes to produce
UINT16 generated;                   // number of bytes generated
BYTE *stream = keyStream;
HMAC_STATE hState;
UINT16 digestSize = CryptHashGetDigestSize(hashAlg);

pAssert(key != NULL && keyStream != NULL);

TEST(TPM_ALG_KDF1_SP800_108);
if (digestSize == 0)
    return 0;
if (counterInOut != NULL)
    counter = *counterInOut;

// If the size of the request is larger than the numbers will handle,
// it is a fatal error.
pAssert(((sizeInBits + 7) / 8) <= INT16_MAX);

// The number of bytes to be generated is the smaller of the sizeInBits bytes or
// the number of requested blocks. The number of blocks is the smaller of the
// number requested or the number allowed by sizeInBits. A partial block is
// a full block.
bytes = (blocks > 0) ? blocks * digestSize : (UINT16)BITS_TO_BYTES(sizeInBits);
generated = bytes;

// Generate required bytes
for (; bytes > 0; bytes -= digestSize)
{
    counter++;
    // Start HMAC
    if (CryptHmacStart(&hState, hashAlg, key->size, key->buffer) == 0)
        return 0;
    // Adding counter
    CryptDigestUpdateInt(&hState.hashState, 4, counter);
    // Adding label
    if (label != NULL)
    {
        HASH_DATA(&hState.hashState, label->size, (BYTE *)label->buffer);
        // Add a null. SP108 is not very clear about when the 0 is needed but to
        // make this like the previous version that did not add an 0x00 after
        // a null-terminated string, this version will only add a null byte
        // if the label parameter did not end in a null byte, or if no label
        // is present.
        if (!((label == NULL) || (label->buffer[label->size - 1] != 0)))
            CryptDigestUpdateInt(&hState.hashState, 1, 0);
        // Adding contextU
        if (contextU != NULL)
            HASH_DATA(&hState.hashState, contextU->size, contextU->buffer);
        // Adding contextV
        if (contextV != NULL)
            HASH_DATA(&hState.hashState, contextV->size, contextV->buffer);
        // Adding size in bits
        CryptDigestUpdateInt(&hState.hashState, 4, sizeInBits);
    }
    // Complete and put the data in the buffer
    CryptHmacEnd(&hState, bytes, stream);
    stream = &stream[digestSize];
}

// Masking in the KDF is disabled. If the calling function wants something
// less than even number of bytes, then the caller should do the masking
// because there is no universal way to do it here
if (counterInOut != NULL)
10.2.13.8.3 CryptKDFe()

KDFe() as defined in TPM specification part 1.

This function returns the number of bytes generated which may be zero.

The Z and keyStream pointers are not allowed to be NULL. The other pointer values may be NULL. The value of sizeInBits must be no larger than \((2^{18})-1 = 256\)K bits (32385 bytes). Any error in the processing of this command is considered fatal.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>hash algorithm is not supported or is TPM_ALG_NULL</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>the number of bytes in the keyStream buffer</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT UINT16 CryptKDFe(
    TPM_ALG_ID hashAlg,    // IN: hash algorithm used in HMAC
    TPM2B *Z,             // IN: Z
    const TPM2B *label,   // IN: a label value for the KDF
    TPM2B *partyUInfo,    // IN: PartyUInfo
    TPM2B *partyVInfo,    // IN: PartyVInfo
    UINT32 sizeInBits,    // IN: size of generated key in bits
    BYTE *keyStream       // OUT: key buffer
) {
    HASH_STATE hashState;
    PHASH_DEF hashDef = CryptGetHashDef(hashAlg);
    UINT32 counter = 0;    // counter value
    UINT16 hLen;
    BYTE *stream = keyStream;
    INT16 bytes;           // number of bytes to generate
    pAssert(keyStream != NULL && Z != NULL && ((sizeInBits + 7) / 8) < INT16_MAX);
    hLen = hashDef->digestSize;
    bytes = (INT16)((sizeInBits + 7) / 8);
    if(hashAlg == TPM_ALG_NULL || bytes == 0)
        return 0;
    // The inner loop of that KDF uses:
    // Hash[i] := H(counter | Z | OtherInfo) (5)
    // Where:
    // Hash[i] the hash generated on the i-th iteration of the loop.
    // H() an approved hash function
    // counter a 32-bit counter that is initialized to 1 and incremented on each iteration
    // Z the X coordinate of the product of a public ECC key and a different private ECC key.
    // OtherInfo a collection of qualifying data for the KDF defined below.
    // In this specification, OtherInfo will be constructed by:
    // OtherInfo := Use | PartyUInfo | PartyVInfo
    for(; bytes > 0; stream = &stream[hLen], bytes = bytes - hLen)
    {
        if(bytes < hLen)
            hlen = bytes;
        counter++;
        // Do the hash
    }
```
CryptHashStart(&hashState, hashAlg);

// Add counter
CryptDigestUpdateInt(&hashState, 4, counter);

// Add Z
if(Z != NULL)
    CryptDigestUpdate2B(&hashState, Z);

// Add label
if(label != NULL)
    CryptDigestUpdate2B(&hashState, label);

// Add a null. SP108 is not very clear about when the 0 is needed but to
// make this like the previous version that did not add an 0x00 after
// a null-terminated string, this version will only add a null byte
// if the label parameter did not end in a null byte, or if no label
// is present.
if((label == NULL)
    || (label->size == 0)
    || (label->buffer[label->size - 1] != 0))
    CryptDigestUpdateInt(&hashState, 1, 0);

// Add PartyUInfo
if(partyUInfo != NULL)
    CryptDigestUpdate2B(&hashState, partyUInfo);

// Add PartyVInfo
if(partyVInfo != NULL)
    CryptDigestUpdate2B(&hashState, partyVInfo);

// Compute Hash. hLen was changed to be the smaller of bytes or hLen
// at the start of each iteration.
CryptHashEnd(&hashState, hLen, stream);

// Mask off bits if the required bits is not a multiple of byte size
if((sizeInBits % 8) != 0)
    keyStream[0] &= ((1 << (sizeInBits % 8)) - 1);

return (UINT16)((sizeInBits + 7) / 8);
10.2.14 CryptHashData.c

```c
#include "Tpm.h"
const HASH_INFO g_hashData[HASH_COUNT + 1] = {
#if ALG_SHA1
    {TPM_ALG_SHA1,    SHA1_DIGEST_SIZE,   SHA1_BLOCK_SIZE,
     SHA1_DER_SIZE,   {SHA1_DER}},
#endif
#if ALG_SHA256
    {TPM_ALG_SHA256,    SHA256_DIGEST_SIZE,   SHA256_BLOCK_SIZE,
     SHA256_DER_SIZE,   {SHA256_DER}},
#endif
#if ALG_SHA384
    {TPM_ALG_SHA384,    SHA384_DIGEST_SIZE,   SHA384_BLOCK_SIZE,
     SHA384_DER_SIZE,   {SHA384_DER}},
#endif
#if ALG_SHA512
    {TPM_ALG_SHA512,    SHA512_DIGEST_SIZE,   SHA512_BLOCK_SIZE,
     SHA512_DER_SIZE,   {SHA512_DER}},
#endif
#if ALG_SM3_256
    {TPM_ALG_SM3_256,    SM3_256_DIGEST_SIZE,   SM3_256_BLOCK_SIZE,
     SM3_256_DER_SIZE,   {SM3_256_DER}},
#endif
    {TPM_ALG_NULL,0,0,0,0}
};
```
10.2.15 CryptPrime.c

10.2.15.1 Introduction

This file contains the code for prime validation.

```c
#include "Tpm.h"
#include "CryptPrime_fp.h"
#include "PrimeTable.h"
#include "CryptPrimeSieve_fp.h"

extern const uint32_t s_LastPrimeInTable;
extern const uint32_t s_PrimeTableSize;
extern const uint32_t s_PrimesInTable;
extern const unsigned char s_PrimeTable[];
extern bigConst s_CompositeOfSmallPrimes;
```

10.2.15.2 Functions

10.2.15.2.1 Root2()

This finds ceil(sqrt(n)) to use as a stopping point for searching the prime table.

```c
static uint32_t Root2(uint32_t n)
{
    int32_t last = (int32_t)(n >> 2);
    int32_t next = (int32_t)(n >> 1);
    int32_t diff;
    int32_t stop = 10;
    // get a starting point
    for(; next != 0; last >>= 1, next >>= 2);
    last++;
    do
    {
        next = (last + (n / last)) >> 1;
        diff = next - last;
        last = next;
        if(stop-- == 0)
            FAIL(FATAL_ERROR_INTERNAL);
    } while(diff < -1 || diff > 1);
    if((n / next) > (unsigned)next)
    {next++;
pAssert(next != 0);
pAssert(((n / next) <= (unsigned)next) && (n / (next + 1) < (unsigned)next));
return next;
    }
```

10.2.15.2.2 IsPrimeInt()

This will do a test of a word of up to 32-bits in size.

```c
BOOL IsPrimeInt(uint32_t n)
{
```

```c
uint32_t i; uint32_t stop;
if(n < 3 || ((n & 1) == 0))
  return (n == 2);
if(n <= s_LastPrimeInTable)
{
  n >>= 1;
  return ((s_PrimeTable[n >> 3] >> (n & 7)) & 1);
}
// Need to search
stop = Root2(n) >> 1;
// starting at 1 is equivalent to staring at (1 << 1) + 1 = 3
for(i = 1; i < stop; i++)
{
  if((s_PrimeTable[i >> 3] >> (i & 7)) & 1)
    // see if this prime evenly divides the number
    if((n % ((i << 1) + 1)) == 0)
      return FALSE;
  }
return TRUE;
}
```

### 10.2.15.2.3 BnIsPrime()

This function is used when the key sieve is not implemented. This function will try to eliminate some of the obvious things before going on to perform MillerRabin() as a final verification of primeness.

```c
BOOL BnIsProbablyPrime(
  bigNum prime,  // IN:
  RAND_STATE *rand  // IN: the random state just
  //     in case Miller-Rabin is required
)
{
  #if RADIX_BITS > 32
  if(BnUnsignedCmpWord(prime, UINT32_MAX) <= 0)
  #else
  if(BnGetSize(prime) == 1)
  #endif
    return IsPrimeInt(prime->d[0]);
  if(BnIsEven(prime))
    return FALSE;
  if(BnUnsignedCmpWord(prime, s_LastPrimeInTable) <= 0)
  {
    crypt_uword_t temp = prime->d[0] >> 1;
    return ((s_PrimeTable[temp >> 3] >> (temp & 7)) & 1);
  }
  { BN_VAR(n, LARGEST_NUMBER_BITS);
    BnGcd(n, prime, s_CompositeOfSmallPrimes);
    if(!BnEqualWord(n, 1))
      return FALSE;
  }
  return MillerRabin(prime, rand);
}
```

### 10.2.15.2.4 MillerRabinRounds()

Function returns the number of Miller-Rabin rounds necessary to give an error probability equal to the security strength of the prime. These values are from FIPS 186-3.

```c
UINT32
```
MillerRabinRounds(
    UINT32           bits)   // IN: Number of bits in the RSA prime
{
    if(bits < 511) return 8;   // don't really expect this
    if(bits < 1536) return 5;  // for 512 and 1K primes
    return 4;                  // for 3K public modulus and greater
}

10.2.15.2.5 MillerRabin()

This function performs a Miller-Rabin test from FIPS 186-3. It does iterations trials on the number. In all likelihood, if the number is not prime, the first test fails.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>probably prime</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>composite</td>
</tr>
</tbody>
</table>

BOOL
MillerRabin(
    bigNum           bnW,
    RAND_STATE      *rand
)
{
    BN_MAX(bnWm1);
    BN_PRIME(bnM);
    BN_PRIME(bnB);
    BN_PRIME(bnZ);
    BOOL             ret = FALSE;   // Assumed composite for easy exit
    unsigned int a;
    unsigned int j;
    int     wLen;
    int     i;
    int     iterations = MillerRabinRounds(BnSizeInBits(bnW));
    //
    // INSTRUMENT_INC(MillerRabinTrials[PrimeIndex]);
    pAssert(bnW->size > 1);
    // Let a be the largest integer such that 2^a divides w1.
    BnSubWord(bnWm1, bnW, 1);
    pAssert(bnWm1->size != 0);
    // Since w is odd (w-1) is even so start at bit number 1 rather than 0
    // Get the number of bits in bnWm1 so that it doesn't have to be recomputed
    // on each iteration.
    i = bnWm1->size * RADIX_BITS;
    // Now find the largest power of 2 that divides w1
    for(a = 1;
        (a < (bnWm1->size * RADIX_BITS)) &
        (BnTestBit(bnWm1, a) == 0);
        a++);
    // 2. m = (w1) / 2^a
    BnShiftRight(bnM, bnWm1, a);
    // 3. wlen = len (w).
    wLen = BnSizeInBits(bnW);
    // 4. For i = 1 to iterations do
    for(i = 0; i < iterations; i++)
    {
        // 4.1 Obtain a string b of wlen bits from an RBG.
        // Ensure that 1 < b < w.
        do
        {
            }
146 BnGetRandomBits(bnB, wLen, rand);
147 // 4.2 If ((b <= 1) or (b >= w1)), then go to step 4.1.
148 } while((BnUnsignedCmpWord(bnB, 1) <= 0)
149 || (BnUnsignedCmp(bnB, bnWm1) >= 0));
150 // 4.3 z = b^m mod w.
151 // if ModExp fails, then say this is not
152 // prime and bail out.
153 BnModExp(bnZ, bnB, bnM, bnW);
154 // 4.4 If ((z == 1) or (z = w == 1)), then go to step 4.7.
155 if((BnUnsignedCmpWord(bnZ, 1) == 0)
156 || (BnUnsignedCmp(bnZ, bnWm1) == 0))
157 goto step4point7;
158 // 4.5 For j = 1 to a 1 do.
159 for(j = 1; j < a; j++)
160 {
161    // 4.5.1 z = z^2 mod w.
162    BnModMult(bnZ, bnZ, bnZ, bnW);
163    // 4.5.2 If (z = w1), then go to step 4.7.
164    if(BnUnsignedCmp(bnZ, bnWm1) == 0)
165       goto step4point7;
166    // 4.5.3 If (z = 1), then go to step 4.6.
167    if(BnEqualWord(bnZ, 1))
168       goto step4point6;
169
170 // 4.6 Return COMPOSITE.
171 step4point6:
172    INSTRUMENT_INC(failedAtIteration[i]);
173    goto end;
175 step4point7:
176    continue;
177
178 // 5. Return PROBABLY PRIME
179 ret = TRUE;
180 end: return ret;
181 }
182 #if !RSA_KEY_SIEVE
183 TPM_RC RsaCheckPrime(
184    bigNum           prime,
185    UINT32           exponent,
186    RAND_STATE      *rand
187 )
188 {
189 #if !RSA_KEY_SIEVE
190 TPM_RC         retVal = TPM_RC_SUCCESS;
191 UINT32          modE = BnModWord(prime, exponent);
192 NOT_REFERENCED(rand);
193 if(modE == 0)
199  // evenly divisible so add two keeping the number odd
200    BnAddWord(prime, prime, 2);
201  // want 0 != (p - 1) mod e
202  // which is 1 != p mod e
203 else if(modE == 1)
204    // subtract 2 keeping number odd and insuring that
205    // 0 != (p - 1) mod e
206    BnSubWord(prime, prime, 2);
207
208  if(BnIsProbablyPrime(prime, rand) == 0)
209    ERROR_RETURN(TPM_RC_VALUE);
210 Exit:
211    return retVal;
212 #else
213    return PrimeSelectWithSieve(prime, exponent, rand);
214 #endif
215 }

10.2.15.2.7 AdjustPrimeCandidate()

This function adjusts the candidate prime so that it is odd and > \sqrt{2}/2. This allows the product of these two numbers to be .5, which, in fixed point notation means that the most significant bit is 1. For this routine, the \sqrt{2}/2 is approximated with 0xB505 which is, in fixed point is 0.7071075439453125 or an error of 0.0001%. Just setting the upper two bits would give a value > 0.75 which is an error of > 6%. Given the amount of time all the other computations take, reducing the error is not much of a cost, but it isn't totally required either.

The function also puts the number on a field boundary.

LIB EXPORT void
RsaAdjustPrimeCandidate(
    bigNum          prime
)
{
    UINT16  highBytes;
    crypt_uword_t   *msw = &prime->d[prime->size - 1];
    #define MASK (MAX_CRYPT_UWORD >> (RADIX_BITS - 16))
    highBytes = *msw >> (RADIX_BITS - 16);
    // This is fixed point arithmetic on 16-bit values
    highBytes = ((UINT32)highBytes * (UINT32)0x4AFB) >> 16;
    highBytes += 0xB505;
    *msw = ((crypt_uword_t)(highBytes) << (RADIX_BITS - 16)) + (*msw & MASK);
    prime->d[0] |= 1;
}

10.2.15.2.8 BnGeneratePrimeForRSA()

Function to generate a prime of the desired size with the proper attributes for an RSA prime.

void
BnGeneratePrimeForRSA(
    bigNum          prime,
    UINT32          bits,
    UINT32          exponent,
    RAND_STATE      *rand
)
{
    BOOL            found = FALSE;
    // Make sure that the prime is large enough
    pAssert(prime->allocated >= BITS_TO_CRYPT_WORDS(bits));
// Only try to handle specific sizes of keys in order to save overhead
pAssert((bits % 32) == 0);
prime->size = BITS_TO_CRYPT_WORDS(bits);
while(!found)
{
    DRBG_Generate(rand, (BYTE *)prime->d, (UINT16)BITS_TO_BYTES(bits));
    RsaAdjustPrimeCandidate(prime);
    found = RsaCheckPrime(prime, exponent, rand) == TPM_RC_SUCCESS;
}
#endif // ALG_RSA
10.2.16 CryptPrimeSieve.c

10.2.16.1 Includes and defines

```c
#include "Tpm.h"
#if RSA_KEY_SIEVE
#include "CryptPrimeSieve_fp.h"
#endif
```

This determines the number of bits in the largest sieve field.

```c
#define MAX_FIELD_SIZE  2048
extern const uint32_t      s_LastPrimeInTable;
extern const uint32_t      s_PrimeTableSize;
extern const uint32_t      s_PrimesInTable;
extern const unsigned char s_PrimeTable[];
```

This table is set of prime markers. Each entry is the prime value for the \( (n + 1) \times 1024 \) prime. That is, the entry in \( s\_PrimeMarkers[1] \) is the value for the 2,048th prime. This is used in the PrimeSieve() to adjust the limit for the prime search. When processing smaller prime candidates, fewer primes are checked directly before going to Miller-Rabin. As the prime grows, it is worth spending more time eliminating primes as, a) the density is lower, and b) the cost of Miller-Rabin is higher.

```c
const uint32_t      s_PrimeMarkersCount = 6;
const uint32_t      s_PrimeMarkers[] = {
  8167, 17881, 28183, 38891, 49871, 60961
};
```

10.2.16.2 Functions

10.2.16.2.1 RsaAdjustPrimeLimit()

This used during the sieve process. The iterator for getting the next prime \( (RsaNextPrime()) \) will return primes until it hits the limit \( (primeLimit) \) set up by this function. This causes the sieve process to stop when an appropriate number of primes have been sieved.

```c
LIB_EXPORT void
RsaAdjustPrimeLimit(
    uint32_t        requestedPrimes )
{
    if(requestedPrimes == 0 || requestedPrimes > s_PrimesInTable)
        requestedPrimes = s_PrimesInTable;
    requestedPrimes = (requestedPrimes - 1) / 1024;
    if(requestedPrimes < s_PrimeMarkersCount)
        primeLimit = s_PrimeMarkers[requestedPrimes];
    else
        primeLimit = s_LastPrimeInTable;
    primeLimit >>= 1;
}
```

10.2.16.2.2 RsaNextPrime()

This the iterator used during the sieve process. The input is the last prime returned (or any starting point) and the output is the next higher prime. The function returns 0 when the \( primeLimit \) is reached.

```c
LIB_EXPORT uint32_t
```
RsaNextPrime(
    uint32_t lastPrime
) {
    if (lastPrime == 0)
        return 0;
    lastPrime >>= 1;
    for (lastPrime += 1; lastPrime <= primeLimit; lastPrime++)
    {
        if (((s_PrimeTable[lastPrime >> 3] >> (lastPrime & 0x7)) & 1) == 1)
            return ((lastPrime << 1) + 1);
    }
    return 0;
}

This table contains a previously sieved table. It has the bits for 3, 5, and 7 removed. Because of the factors, it needs to be aligned to 105 and has a repeat of 105.

const BYTE seedValues[] = {
0x16, 0x29, 0xcb, 0xa4, 0x65, 0xda, 0x30, 0x6c,
0x99, 0x96, 0x4c, 0x53, 0xa2, 0x2d, 0x52, 0x96,
0x49, 0xcb, 0xb4, 0x61, 0xd8, 0x32, 0x2d, 0x99,
0x48, 0x59, 0x26, 0x6d, 0x91, 0xb2, 0x4c,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x1b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
0x25, 0xda, 0x22, 0x65, 0x99, 0xb4, 0x4c, 0x4b,
91 0x04, 0x05, 0x06, 0x06, 0x06, 0x06, 0x07, 0x004, 0x05, 0x05, 0x06, 0x06, 0x06, 0x07, 0x06, 0x06, 0x07, 0x007, 0x06, 0x07, 0x07, 0x08, 0x05, 0x06, 0x06, 0x07, 0x06, 0x07, 0x07, 0x08, 0x07, 0x08, 0x08, 0x09, 0x09, 0x0a, 0x0a, 0x0b, 0x0b, 0x0c, 0x0c, 0x0d, 0x0d, 0x0e, 0x0e, 0x0f, 0x0f}
92 #define BitsInByte(x) bitsInByte[(unsigned char)x]
93 #else
94 const BYTE bitsInNibble[16] = {
95 0x00, 0x01, 0x01, 0x02, 0x01, 0x02, 0x02, 0x03,
96 0x01, 0x02, 0x02, 0x03, 0x02, 0x03, 0x03, 0x04};
97 #define BitsInByte(x) \
98 (bitsInNibble[(unsigned char)(x) & 0xf] \
99 + bitsInNibble[((unsigned char)(x) >> 4) & 0xf])
100 #endif

10.2.16.2.3 BitsInArray()

This function counts the number of bits set in an array of bytes.

104 static int
105 BitsInArray(
106   const unsigned char *a, // IN: A pointer to an array of bytes
107   unsigned int aSize   // IN: the number of bytes to sum
108 )
109 {
110   int j = 0;
111   for(; aSize; a++, aSize--)
112     j += BitsInByte(*a);
113   return j;
114 }

10.2.16.2.4 FindNthSetBit()

This function finds the nth SET bit in a bit array. The n parameter is between 1 and the number of bits in the array (always a multiple of 8). If called when the array does not have n bits set, it will return -1.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;0</td>
<td>no bit is set or no bit with the requested number is set</td>
</tr>
<tr>
<td>&gt;=0</td>
<td>the number of the bit in the array that is the nth set</td>
</tr>
</tbody>
</table>

115 LIB_EXPORT int
116 FindNthSetBit(
117   const UINT16 aSize, // IN: the size of the array to check
118   const BYTE   *a,   // IN: the array to check
119   const UINT32 n    // IN, the number of the SET bit
120 )
121 {
122   UINT16 i;
123   int  retValue;
124   UINT32 sum = 0;
125   BYTE  sel;
126   //find the bit
127   for(i = 0; (i < (int)aSize) && (sum < n); i++)
128     sum += BitsInByte(a[i]);
129   i--;
130   // The chosen bit is in the byte that was just accessed
131   // Compute the offset to the start of that byte
132   // Subtract the bits in the last byte added.
133   retValue = i * 8 - 1;
134   sel = a[i];
135   sum -= BitsInByte(sel);
// Now process the byte, one bit at a time.
for(; (sel != 0) && (sum != n); retValue++, sel = sel >> 1)
    sum += (sel & 1) != 0;
return (sum == n) ? retValue : -1;

typedef struct
{
    UINT16    prime;
    UINT16    count;
} SIEVE_MARKS;
const SIEVE_MARKS sieveMarks[5] = {
{31, 7}, {73, 5}, {241, 4}, {1621, 3}, {UINT16_MAX, 2}};

10.2.16.2.5 PrimeSieve()

This function does a prime sieve over the input field which has as its starting address the value in bnN. Since this initializes the Sieve using a precomputed field with the bits associated with 3, 5 and 7 already turned off, the value of pnN may need to be adjusted by a few counts to allow the precomputed field to be used without modification.

To get better performance, one could address the issue of developing the composite numbers. When the size of the prime gets large, the time for doing the divisions goes up, noticeably. It could be better to develop larger composite numbers even if they need to be bigNum's themselves. The object would be to reduce the number of times that the large prime is divided into a few large divides and then use smaller divides to get to the final 16 bit (or smaller) remainders.

LIB_EXPORT UINT32
PrimeSieve(
    bigNum           bnN,          // IN/OUT: number to sieve
    UINT32           fieldSize,   // IN: size of the field area in bytes
    BYTE            *field        // IN: field
    )
{
    INT32            i;
    UINT32           j;
    UINT32           fieldBits = fieldSize * 8;
    UINT32           r;
    BYTE            *pField;
    INT32            iter;
    INT32            adjust;
    INT32            mark = 0;
    INT32            count = sieveMarks[0].count;
    INT32            stop = sieveMarks[0].prime;
    INT32            composite;
    UINT32           pList[8];
    INT32            next;

    pAssert(field != NULL && bnN != NULL);
    if(BnModWord(bnN, 105) == 0)
        adjust = 105;
    else
        adjust = BnModWord(bnN, 105);
    // Adjust the input number so that it points to the first number in a
    // aligned field.
    BnSubWord(bnN, bnN, adjust);
    pField = field;
    for(i = fieldSize; i >= sizeof(seedValues); i -= sizeof(seedValues))
        pField += sizeof(seedValues);


memcpy(pField, seedValues, sizeof(seedValues));

if(i != 0)
    memcpy(pField, seedValues, i);

// Cycle through the primes, clearing bits
// Have already done 3, 5, and 7
iter = 7;

#define NEXT_PRIME(iter) (iter = RsaNextPrime(iter))

// Get the next N primes where N is determined by the mark in the sieveMarks
while((composite = NEXT_PRIME(iter)) != 0)
{
    next = 0;
    i = count;
    for(; i > 0; i--)
    {
        next = NEXT_PRIME(iter);
        pList[i] = next;
        if(next != 0)
            composite *= next;
    }

    // Get the remainder when dividing the base field address
    // by the composite
    composite = BnModWord(bnN, composite);

    // 'composite' is divisible by the composite components. for each of the
    // composite components, divide 'composite'. That remainder (r) is used to
    // pick a starting point for clearing the array. The stride is equal to the
    // composite component. Note, the field only contains odd numbers. If the
    // field were expanded to contain all numbers, then half of the bits would
    // have already been cleared. We can save the trouble of clearing them a
    // second time by having a stride of 2*next. Or we can take all of the even
    // numbers out of the field and use a stride of 'next'
    for(i = count; i > 0; i--)
    {
        next = pList[i];
        if(next == 0)
            goto done;
        r = composite % next;
        // these computations deal with the fact that the field starts at some
        // arbitrary offset within the number space. If the field were all numbers,
        // then we would have gone through some number of bit clearings before we
        // got to the start of this range. We don't know how many there were before,
        // but we can tell from the remainder whether we are on an even or odd
        // stride when we hit the beginning of the table. If we are on an odd stride
        // (r & 1), we would start half a stride in (next - r)/2. If we are on an
        // even stride, we need 1.5 strides (next + r/2) because the table only has
        // odd numbers. If the remainder happens to be zero, then the start of the
        // table is on stride so no adjustment is necessary.
        if(r & 1)
            j = (next - r) / 2;
        else if(r == 0)
            j = 0;
        else
            j = next - r / 2;
        for(; j < fieldBits; j += next)
            ClearBit(j, field, fieldSize);
    }

    if(next >= stop)
    {
        mark++;
        count = sieveMarks[mark].count;
        stop = sieveMarks[mark].prime;
    }
}

done:
    INSTRUMENT_INC(totalFieldsSieved[PrimeIndex]);
252     i = BitsInArray(field, fieldSize);
253     INSTRUMENT_ADD(bitsInFieldAfterSieve[PrimeIndex], i);
254     INSTRUMENT_ADD(emptyFieldsSieved[PrimeIndex], (i == 0));
255     return i;
256 } #ifdef SIEVE_DEBUG
257 static uint32_t fieldSize = 210;

10.2.16.2.6 SetFieldSize()

Function to set the field size used for prime generation. Used for tuning.

259 LIB_EXPORT uint32_t
260 SetFieldSize(
261     uint32_t         newFieldSize
262 )
263 {
264     if(newFieldSize == 0 || newFieldSize > MAX_FIELD_SIZE)
265         fieldSize = MAX_FIELD_SIZE;
266     else
267         fieldSize = newFieldSize;
268     return fieldSize;
269 } #endif // SIEVE_DEBUG

10.2.16.2.7 PrimeSelectWithSieve()

This function will sieve the field around the input prime candidate. If the sieve field is not empty, one of the one bits in the field is chosen for testing with Miller-Rabin. If the value is prime, \( p_{nP} \) is updated with this value and the function returns success. If this value is not prime, another pseudo-random candidate is chosen and tested. This process repeats until all values in the field have been checked. If all bits in the field have been checked and none is prime, the function returns FALSE and a new random value needs to be chosen.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SUCCESS</td>
<td>candidate is probably prime</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>candidate is not prime and couldn't find and alternative in the field</td>
</tr>
</tbody>
</table>

271 LIB_EXPORT TPM_RC
272 PrimeSelectWithSieve(
273     bigNum           candidate,       // IN/OUT: The candidate to filter
274     UINT32           e,               // IN: the exponent
275     RAND_STATE      *rand             // IN: the random number generator state
276 )
277 {
278     BYTE             field[MAX_FIELD_SIZE];
279     UINT32           first;
280     UINT32           ones;
281     INT32            chosen;
282     BN_PRIME(test);    
283     UINT32           modE;
284     #ifndef SIEVE_DEBUG
285     UINT32           fieldSize = MAX_FIELD_SIZE;
286     #endif
287     UINT32           primeSize;
288     //
289     // Adjust the field size and prime table list to fit the size of the prime
290     // being tested. This is done to try to optimize the trade-off between the
291     // dividing done for sieving and the time for Miller-Rabin. When the size
292     // of the prime is large, the cost of Miller-Rabin is fairly high, as is the
// cost of the sieving. However, the time for Miller-Rabin goes up considerably
// faster than the cost of dividing by a number of primes.
primeSize = BnSizeInBits(candidate);

if(primeSize <= 512)
    { // Use just the first 1024 primes
        RsaAdjustPrimeLimit(1024);
    }
else if(primeSize <= 1024)
    { // Use just the first 4K primes
        RsaAdjustPrimeLimit(4096);
    }
else
    { // Use all available
        RsaAdjustPrimeLimit(0);
    }

// Save the low-order word to use as a search generator and make sure that
// it has some interesting range to it
first = candidate->d[0] | 0x80000000;

// Sieve the field
ones = PrimeSieve(candidate, fieldSize, field);
pAssert(ones > 0 && ones < (fieldSize * 8));
for(; ones > 0; ones--)
    {
        // Decide which bit to look at and find its offset
        chosen = FindNthSetBit((UINT16)fieldSize, field, ((first % ones) + 1));
        if((chosen < 0) || (chosen >= (INT32)(fieldSize * 8)))
            FAIL(FATAL_ERROR_INTERNAL);

        // Set this as the trial prime
        BnAddWord(test, candidate, (crypt_uword_t)(chosen * 2));

        // The exponent might not have been one of the tested primes so
        // make sure that it isn't divisible and make sure that 0 != (p-1) mod e
        // Note: This is the same as 1 != p mod e
        modE = BnModWord(test, e);
        if((modE != 0) && (modE != 1) && MillerRabin(test, rand))
            { // Run out of bits and couldn't find a prime in this field
                INSTRUMENT_INC(noPrimeFields[PrimeIndex]);
                return TPM_RC_NO_RESULT;
            }
        // Clear the bit just tested
        ClearBit(chosen, field, fieldSize);
    }

// Clear value
#define CLEAR_VALUE(x) memset(x, 0, sizeof(x))

void
RsaSimulationEnd

#if RSA_INSTRUMENT
static char a[256];
char *
PrintTuple(
    UINT32 *i
)
    {
        sprintf(a, "{%d, %d, %d}", i[0], i[1], i[2]);
        return a;
    }
#endif
void
)
{
    int        i;
    UINT32    averages[3];
    UINT32    nonFirst = 0;
    if((PrimeCounts[0] + PrimeCounts[1] + PrimeCounts[2]) != 0)
    {
        printf("Primes generated = %s\n", PrintTuple(PrimeCounts));
        printf("Fields sieved = %s\n", PrintTuple(totalFieldsSieved));
        printf("Fields with no primes = %s\n", PrintTuple(noPrimeFields));
        printf("Primes checked with Miller-Rabin = %s\n",
               PrintTuple(MillerRabinTrials));
        for(i = 0; i < 3; i++)
            averages[i] = (totalFieldsSieved[i] != 0 ? bitsInFieldAfterSieve[i] / totalFieldsSieved[i] : 0);
        printf("Average candidates in field %s\n", PrintTuple(averages));
        for(i = 1; i < (sizeof(failedAtIteration) / sizeof(failedAtIteration[0])); i++)
            nonFirst += failedAtIteration[i];
        printf("Miller-Rabin failures not in first round = %d\n", nonFirst);
    }
    CLEAR_VALUE(PrimeCounts);
    CLEAR_VALUE(totalFieldsSieved);
    CLEAR_VALUE(noPrimeFields);
    CLEAR_VALUE(MillerRabinTrials);
    CLEAR_VALUE(bitsInFieldAfterSieve);
}

LIB_EXPORT void
GetSieveStats(
    uint32_t    *trials,
    uint32_t    *emptyFields,
    uint32_t    *averageBits
)
{
    uint32_t    totalBits;
    uint32_t    fields;
    *emptyFields = noPrimeFields[0] + noPrimeFields[1] + noPrimeFields[2];
    fields = totalFieldsSieved[0] + totalFieldsSieved[1]
    + totalFieldsSieved[2];
    totalBits = bitsInFieldAfterSieve[0] + bitsInFieldAfterSieve[1]
    + bitsInFieldAfterSieve[2];
    if(fields != 0)
        *averageBits = totalBits / fields;
    else
        *averageBits = 0;
    CLEAR_VALUE(PrimeCounts);
    CLEAR_VALUE(totalFieldsSieved);
    CLEAR_VALUE(noPrimeFields);
    CLEAR_VALUE(MillerRabinTrials);
    CLEAR_VALUE(bitsInFieldAfterSieve);
}
#endif
#endif
// RSA_KEY_SIEVE
#if !RSA_INSTRUMENT
void
RsaSimulationEnd(
    void
)
#endif
425   {
426   }
427  #endif
10.2.17 CryptRand.c

10.2.17.1 Introduction

This file implements a DRBG with a behavior according to SP800-90A using a block cypher. This is also compliant to ISO/IEC 18031:2011(E) C.3.2.

A state structure is created for use by TPM.lib and functions within the CryptoEngine() my use their own state structures when they need to have deterministic values.

A debug mode is available that allows the random numbers generated for TPM.lib to be repeated during runs of the simulator. The switch for it is in TpmBuildSwitches.h. It is USE_DEBUG_RNG.

This is the implementation layer of CTR DRGB mechanism as defined in SP800-90A and the functions are organized as closely as practical to the organization in SP800-90A. It is intended to be compiled as a separate module that is linked with a secure application so that both reside inside the same boundary [SP 800-90A 8.5]. The secure application in particular manages the accesses protected storage for the state of the DRBG instantiations, and supplies the implementation functions here with a valid pointer to the working state of the given instantiations (as a DRBG_STATE structure).

This DRBG mechanism implementation does not support prediction resistance. Thus prediction_resistance_flag is omitted from Instantiate_function(), Reseed_function(), Generate_function() argument lists [SP 800-90A 9.1, 9.2, 9.3], as well as from the working state data structure DRBG_STATE [SP 800-90A 9.1].

This DRBG mechanism implementation always uses the highest security strength of available in the block ciphers. Thus requested_security_strength parameter is omitted from Instantiate_function() and Generate_function() argument lists [SP 800-90A 9.1, 9.2, 9.3], as well as from the working state data structure DRBG_STATE [SP 800-90A 9.1].

Internal functions (ones without Crypt prefix) expect validated arguments and therefore use assertions instead of runtime parameter checks and mostly return void instead of a status value.

```c
#include "Tpm.h"

Pull in the test vector definitions and define the space

#include "PRNG_TestVectors.h"
const BYTE DRBG_NistTestVector_Entropy[] = {DRBG_TEST_INITIATE_ENTROPY};
const BYTE DRBG_NistTestVector_GeneratedInterm[] =
    {DRBG_TEST_GENERATED_INTERM};
const BYTE DRBG_NistTestVector_EntropyReseed[] =
    {DRBG_TEST_RESEED_ENTROPY};
const BYTE DRBG_NistTestVector_Generated[] = {DRBG_TEST_GENERATED};
```

10.2.17.2 Derivation Functions

10.2.17.2.1 Description

The functions in this section are used to reduce the personalization input values to make them usable as input for reseeding and instantiation. The overall behavior is intended to produce the same results as described in SP800-90A, section 10.4.2 "Derivation Function Using a Block Cipher Algorithm (Block_Cipher_df())." The code is broken into several subroutines to deal with the fact that the data used for personalization may come in several separate blocks such as a Template hash and a proof value and a primary seed.
10.2.17.2.2 Derivation Function Defines and Structures

```
#define DF_COUNT (DRBG_KEY_SIZE_WORDS / DRBG_IV_SIZE_WORDS + 1)
#if DRBG_KEY_SIZE_BITS != 128 && DRBG_KEY_SIZE_BITS != 256
  #error "CryptRand.c only written for AES with 128- or 256-bit keys."
#endif
typedef struct
  {
    DRBG_KEY_SCHEDULE   keySchedule;
    DRBG_IV             iv[DF_COUNT];
    DRBG_IV             out1;
    DRBG_IV             buf;
    int                 contents;
  } DF_STATE, *PDF_STATE;
```

10.2.17.2.3 DfCompute()

This function does the incremental update of the derivation function state. It encrypts the iv value and XOR's the results into each of the blocks of the output. This is equivalent to processing all of input data for each output block.

```
static void DfCompute(
    PDF_STATE        dfState
)
{
    int i;
    int iv;
    crypt_uword_t *pIv;
    crypt_uword_t temp[DRBG_IV_SIZE_WORDS] = {0};
    //
    for(iv = 0; iv < DF_COUNT; iv++)
    {
        pIv = (crypt_uword_t *)&dfState->iv[iv].words[0];
        for(i = 0; i < DRBG_IV_SIZE_WORDS; i++)
        {
            temp[i] ^= pIv[i] ^ dfState->buf.words[i];
        }
    }
    DRBG_ENCRYPT(&dfState->keySchedule, &temp, pIv);
    for(i = 0; i < DRBG_IV_SIZE_WORDS; i++)
    dfState->buf.words[i] = 0;
    dfState->contents = 0;
}
```

10.2.17.2.4 DfStart()

This initializes the output blocks with an encrypted counter value and initializes the key schedule.

```
static void DfStart(
    PDF_STATE        dfState,
    uint32_t         inputLength
)
{
    BYTE init[8];
    int i;
    UINT32 drbgSeedSize = sizeof(DRBG_SEED);
    const BYTE dfKey[DRBG_KEY_SIZE_BYTES] = {
      0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
      0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f
```
#if DRBG_KEY_SIZE_BYTES > 16
    ,0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, 0x17,
    0x18, 0x19, 0x1a, 0x1b, 0x1c, 0x1d, 0x1e, 0x1f
#endif

memcpy(dfState, 0, sizeof(DF_STATE));
DRBG_ENCRYPT_SETUP(&dfKey[0], DRBG_KEY_SIZE_BITS, &dfState->keySchedule);

// Create the first chaining values
for(i = 0; i < DF_COUNT; i++)
    ((BYTE *)&dfState->iv[i])[3] = (BYTE)i;
DfCompute(dfState);

// initialize the first 64 bits of the IV in a way that doesn't depend
// on the size of the words used.
UINT32_TO_BYTE_ARRAY(inputLength, init);
UINT32_TO_BYTE_ARRAY(drbgSeedSize, &init[4]);
memcpy(&dfState->iv[0], init, 8);
dfState->contents = 4;

}

10.2.17.2.5 DfUpdate()

This updates the state with the input data. A byte at a time is moved into the state buffer until it is full and then that block is encrypted by DfCompute().

static void
DfUpdate(
    PDF_STATE        dfState,
    int              size,
    const BYTE      *data
)
{
    while(size > 0)
    {
        int toFill = DRBG_IV_SIZE_BYTES - dfState->contents;
        if(size < toFill)
            toFill = size;
        // Copy as many bytes as there are or until the state buffer is full
        memcpy(&dfState->buf.bytes[dfState->contents], data, toFill);
        // Reduce the size left by the amount copied
        size -= toFill;
        // Advance the data pointer by the amount copied
        data += toFill;
        // increase the buffer contents count by the amount copied
        dfState->contents += toFill;
        pAssert(dfState->contents <= DRBG_IV_SIZE_BYTES);
        // If we have a full buffer, do a computation pass.
        if(dfState->contents == DRBG_IV_SIZE_BYTES)
            DfCompute(dfState);
    }
}

10.2.17.2.6 DfEnd()

This function is called to get the result of the derivation function computation. If the buffer is not full, it is padded with zeros. The output buffer is structured to be the same as a DRBG_SEED value so that the function can return a pointer to the DRBG_SEED value in the DF_STATE structure.

static DRBG_SEED *
DfEnd(
    PDF_STATE        dfState
)
{
106 // Since DfCompute is always called when a buffer is full, there is always
107 // space in the buffer for the terminator
108 dfState->buf.bytes[dfState->contents++] = 0x80;
109 // If the buffer is not full, pad with zeros
110 while(dfState->contents < DRBG_IV_SIZE_BYTES)
111    dfState->buf.bytes[dfState->contents++] = 0;
112 // Do a final state update
113 DfCompute(dfState);
114 return (DRBG_SEED *)&dfState->iv;
115 }

10.2.17.2.7 DfBuffer()

Function to take an input buffer and do the derivation function to produce a DRBG_SEED value that can
be used in DRBG_Reseed();

116 static DRBG_SEED *
117 DfBuffer(
118     DRBG_SEED       *output,       // OUT: receives the result
119     int            size,          // IN: size of the buffer to add
120     BYTE           *buf            // IN: address of the buffer
121 )
122 {
123     DF_STATE        dfState;
124     if(size == 0 || buf == NULL)
125         return NULL;
126     // Initialize the derivation function
127     DfStart(&dfState, size);
128     DfUpdate(&dfState, size, buf);
129     DfEnd(&dfState);
130     memcpy(output, &dfState.iv[0], sizeof(DRBG_SEED));
131     return output;
132 }

10.2.17.2.8 DRBG_GetEntropy()

Even though this implementation never fails, it may get blocked indefinitely long in the call to get entropy
from the platform (DRBG_GetEntropy32()). This function is only used during instantiation of the DRBG for
manufacturing and on each start-up after an non-orderly shutdown.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>requested entropy returned</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>entropy Failure</td>
</tr>
</tbody>
</table>

133 BOOL
134 DRBG_GetEntropy(
135     UINT32           requiredEntropy,       // IN: requested number of bytes of full
136                     // entropy
137     BYTE           *entropy            // OUT: buffer to return collected entropy
138 )
139 {
140 #if !USE_DEBUG_RNG
141    UINT32         obtainedEntropy;
142    INT32          returnedEntropy;
143 // If in debug mode, always use the self-test values for initialization
144    if(IsSelfTest())
145    {
146 #endif

Page 644
149 // If doing simulated DRBG, then check to see if the
150 // entropyFailure condition is being tested
151 if(!IsEntropyBad())
152 {
153 // In self-test, the caller should be asking for exactly the seed
154 // size of entropy.
155 pAssert(requiredEntropy == sizeof(DRBG_NistTestVector_Entropy));
156 memcpy(entropy, DRBG_NistTestVector_Entropy,
157 sizeof(DRBG_NistTestVector_Entropy));
158 }
159 #if !USE_DEBUG_RNG
160 }
161 else if(!IsEntropyBad())
162 {
163 // Collect entropy
164 // Note: In debug mode, the only "entropy" value ever returned
165 // is the value of the self-test vector.
166 for(returnedEntropy = 1, obtainedEntropy = 0;
167 obtainedEntropy < requiredEntropy && !IsEntropyBad();
168 obtainedEntropy += returnedEntropy)
169 {
170 returnedEntropy = _plat__GetEntropy(entropy[obtainedEntropy],
171 requiredEntropy - obtainedEntropy);
172 if(returnedEntropy <= 0)
173 SetEntropyBad();
174 }
175 #endif
176 return !IsEntropyBad();
177 }
178 }

10.2.17.2.9 IncrementIv()

Used by EncryptDRBG()

179 void
180 IncrementIv(
181 DRBG_IV         *iv
182 )
183 {
184 BYTE      *ivP = ((BYTE *)iv) + DRBG_IV_SIZE_BYTES;
185 while((--ivP >= (BYTE *)iv) && (*ivP = ((*ivP + 1) & 0xFF)) == 0));
186 }

10.2.17.2.10 EncryptDRBG()

This does the encryption operation for the DRBG. It will encrypt the input state counter (IV) using the state key. Into the output buffer for as many times as it takes to generate the required number of bytes.

187 void
188 EncryptDRBG(
189 BYTE                *dOut,
190 UINT32               dOutBytes,
191 DRBG_KEY_SCHEDULE   *keySchedule,
192 DRBG_IV             *iv,
193 UINT32              *lastValue // Points to the last output value
194 )
195 {
196 #if FIPS_COMPLIANT
197 // For FIPS compliance, the DRBG has to do a continuous self-test to make sure that
198 // no two consecutive values are the same. This overhead is not incurred if the TPM
199 // is not required to be FIPS compliant
200 //
Trusted Platform Module Library

Part 4: Supporting Routines

Page 646

TCG Public Review

Family “2.0”

September 18, 2018

Copyright © TCG 2006-2019

Level 00 Revision 01.50

UINT32 temp[DRBG_IV_SIZE_BYTES / sizeof(UINT32)];

int i;

BYTE *p;

for(; dOutBytes > 0;)
{
    // Increment the IV before each encryption (this is what makes this
    // different from normal counter-mode encryption
    IncrementIv(iv);
    DBRG_ENCRYPT(keySchedule, iv, temp);
    // Expect a 16 byte block
    #if DRBG_IV_SIZE_BITS != 128
        #error "Unsupported IV size in DRBG"
    #endif
    if((lastValue[0] == temp[0])
        && (lastValue[1] == temp[1])
        && (lastValue[2] == temp[2])
        && (lastValue[3] == temp[3])
    )
        FAIL(FATAL_ERROR_DRBG);
    lastValue[0] = temp[0];
    lastValue[1] = temp[1];
    lastValue[2] = temp[2];
    lastValue[3] = temp[3];
    i = MIN(dOutBytes, DRBG_IV_SIZE_BYTES);
    dOutBytes -= i;
    for(p = (BYTE *)temp; i > 0; i--)
        *dOut++ = *p++;
}
#else
    // version without continuous self-test
    NOT_REFERENCED(lastValue);
    for(; dOutBytes >= DRBG_IV_SIZE_BYTES;
        dOut = &dOut[DRBG_IV_SIZE_BYTES], dOutBytes -= DRBG_IV_SIZE_BYTES)
    {
        // Increment the IV
        IncrementIv(iv);
        DBRG_ENCRYPT(keySchedule, iv, dOut);)
    // If there is a partial, generate into a block-sized
    // temp buffer and copy to the output.
    if(dOutBytes != 0)
    {
        BYTE temp[DRBG_IV_SIZE_BYTES];
        // Increment the IV
        IncrementIv(iv);
        DBRG_ENCRYPT(keySchedule, iv, temp);
        memcpy(dOut, temp, dOutBytes);
    }
#endif
}

10.2.17.2.11 DRBG_Update()

This function performs the state update function. According to SP800-90A, a temp value is created by
doing CTR mode encryption of providedData and replacing the key and IV with these values. The one
difference is that, with counter mode, the IV is incremented after each block is encrypted and in this
operation, the counter is incremented before each block is encrypted. This function implements an
optimized version of the algorithm in that it does the update of the drbgState->seed in place and then
providedData is XORed() into drbgState->seed to complete the encryption of providedData. This works
because the IV is the last thing that gets encrypted.

void DRBG_Update(

}
```
DRBG_STATE *drbgState, // IN:OUT state to update
DRBG_KEY_SCHEDULE *keySchedule, // IN: the key schedule (optional)
DRBG_SEED *providedData // IN: additional data
}

UINT32 i;
BYTE *temp = (BYTE *)&drbgState->seed;
DRBG_KEY *key = pDRBG_KEY(&drbgState->seed);
DRBG_IV *iv = pDRBG_IV(&drbgState->seed);
DRBG_KEY_SCHEDULE localKeySchedule;
// pAssert(drbgState->magic == DRBG_MAGIC);

// If an key schedule was not provided, make one
if(keySchedule == NULL)
{
    if(DRBG_ENCRYPT_SETUP((BYTE *)key,
        DRBG_KEY_SIZE_BITS, &localKeySchedule) != 0)
    {
        FAIL(FATAL_ERROR_INTERNAL);
        keySchedule = &localKeySchedule;
    }
    // Encrypt the temp value
    EncryptDRBG(temp, sizeof(DRBG_SEED), keySchedule, iv,
        drbgState->lastValue);
    if(providedData != NULL)
    {
        BYTE *pP = (BYTE *)providedData;
        for(i = DRBG_SEED_SIZE_BYTES; i != 0; i--)
        {
            *temp++ ^= *pP++;
        }
    }
    // Since temp points to the input key and IV, we are done and
    // don't need to copy the resulting 'temp' to drbgState->seed
}

10.2.17.2.12 DRBG_Reseed()

This function is used when reseeding of the DRBG is required. If entropy is provided, it is used in lieu of
using hardware entropy.

NOTE: the provided entropy must be the required size.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>reseed succeeded</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>reseed failed, probably due to the entropy generation</td>
</tr>
</tbody>
</table>

BOOL  
DRBG_Reseed(
    DRBG_STATE *drbgState, // IN: the state to update
    DRBG_SEED *providedEntropy, // IN: entropy
    DRBG_SEED *additionalData // IN:
)
{
    DRBG_SEED seed;
    pAssert((drbgState != NULL) && (drbgState->magic == DRBG_MAGIC));
    if(providedEntropy == NULL)
    {
        providedEntropy = &seed;
        if(!DRBG_GetEntropy(sizeof(DRBG_SEED), (BYTE *)providedEntropy))
            return FALSE;
    }
```

```
Family "2.0"  
TCG Public Review  
Level 00 Revision 01.50  
Copyright © TCG 2006-2019  
September 18, 2018
if (additionalData != NULL) {
    unsigned int i;
    // XOR the provided data into the provided entropy
    for (i = 0; i < sizeof(DRBG_SEED); i++) {
        ((BYTE *)providedEntropy)[i] ^= ((BYTE *)additionalData)[i];
    }
    DRBG_Update(drbgState, NULL, providedEntropy);
    drbgState->reseedCounter = 1;
    return TRUE;
}

10.2.17.2.13 DRBG_SelfTest()

This is run when the DRBG is instantiated and at startup

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>test OK</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>test failed</td>
</tr>
</tbody>
</table>

BOOL DRBG_SelfTest(
    void
) {
    BYTE buf[sizeof(DRBG_NistTestVector_Generated)];
    DRBG_SEED seed;
    UINT32 i;
    BYTE *p;
    DRBG_STATE testState;
    // pAssert(!IsSelfTest());
    SetSelfTest();
    SetDrbgTested();
    // Do an instantiate
    if (!DRBG_Instantiate(&testState, 0, NULL))
        return FALSE;
    #if DRBG_DEBUG_PRINT
dbgDumpMemBlock(pDRBG_KEY(&testState), DRBG_KEY_SIZE_BYTES,
        "Key after Instantiate");
dbgDumpMemBlock(pDRBG_IV(&testState), DRBG_IV_SIZE_BYTES,
        "Value after Instantiate");
    #endif
    if (DRBG_Generate((RAND_STATE *)&testState, buf, sizeof(buf)) == 0)
        return FALSE;
    #if DRBG_DEBUG_PRINT
dbgDumpMemBlock(pDRBG_KEY(&testState.seed), DRBG_KEY_SIZE_BYTES,
        "Key after 1st Generate");
dbgDumpMemBlock(pDRBG_IV(&testState.seed), DRBG_IV_SIZE_BYTES,
        "Value after 1st Generate");
    #endif
    if (memcmp(buf, DRBG_NistTestVector_GeneratedInterm, sizeof(buf)) != 0)
        return FALSE;
    memcpy(seed.bytes, DRBG_NistTestVector_EntropyReseed, sizeof(seed));
    DRBG_Reseed(&testState, &seed, NULL);
    #if DRBG_DEBUG_PRINT
dbgDumpMemBlock((BYTE *)pDRBG_KEY(&testState.seed), DRBG_KEY_SIZE_BYTES,
    "Key after Reseed");
dbgDumpMemBlock(pDRBG_IV(&testState.seed), DRBG_IV_SIZE_BYTES,
        "Value after Reseed");
    #endif
    if (memcmp(buf, DRBG_NistTestVector_GeneratedInterm, sizeof(buf)) != 0)
        return FALSE;
    memcpy(seed.bytes, DRBG_NistTestVector_EntropyReseed, sizeof(seed));
    DRBG_Reseed(&testState, &seed, NULL);
    #if DRBG_DEBUG_PRINT
dbgDumpMemBlock((BYTE *)pDRBG_KEY(&testState.seed), DRBG_KEY_SIZE_BYTES,
    "Key after Reseed");
dbgDumpMemBlock(pDRBG_IV(&testState.seed), DRBG_IV_SIZE_BYTES,
        "Value after Reseed");
    #endif
    if (memcmp(buf, DRBG_NistTestVector_GeneratedInterm, sizeof(buf)) != 0)
        return FALSE;
    memcpy(seed.bytes, DRBG_NistTestVector_EntropyReseed, sizeof(seed));
    DRBG_Reseed(&testState, &seed, NULL);
    #if DRBG_DEBUG_PRINT
dbgDumpMemBlock((BYTE *)pDRBG_KEY(&testState.seed), DRBG_KEY_SIZE_BYTES,
    "Key after Reseed");
dbgDumpMemBlock(pDRBG_IV(&testState.seed), DRBG_IV_SIZE_BYTES,
        "Value after Reseed");
    #endif
    if (memcmp(buf, DRBG_NistTestVector_GeneratedInterm, sizeof(buf)) != 0)
        return FALSE;
    memcpy(seed.bytes, DRBG_NistTestVector_EntropyReseed, sizeof(seed));
    DRBG_Reseed(&testState, &seed, NULL);
    #if DRBG_DEBUG_PRINT
dbgDumpMemBlock((BYTE *)pDRBG_KEY(&testState.seed), DRBG_KEY_SIZE_BYTES,
    "Key after Reseed");
dbgDumpMemBlock(pDRBG_IV(&testState.seed), DRBG_IV_SIZE_BYTES,
        "Value after Reseed");
    #endif
    if (memcmp(buf, DRBG_NistTestVector_GeneratedInterm, sizeof(buf)) != 0)
        return FALSE;
    memcpy(seed bytes, DRBG_NistTestVector_EntropyReseed, sizeof(seed));
    DRBG_Reseed(&test State, &seed, NULL);
10.2.17.3 Public Interface

10.2.17.3.1 Description

The functions in this section are the interface to the RNG. These are the functions that are used by TPM.lib. Other functions are only visible to programs in the LtcCryptoEngine().

10.2.17.3.2 CryptRandomStir()

This function is used to cause a reseed. A DRBG_SEED amount of entropy is collected from the hardware and then additional data is added.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>failure of the entropy generator</td>
</tr>
</tbody>
</table>

LIB_EXPORT TPM_RC
CryptRandomStir(
  UINT16 additionalDataSize,
  BYTE *additionalData)

#if !USE_DEBUG_RNG
  DBRG_SEED _ tmpBuf;
  DBRG_SEED _ dfResult;

  // All reseed with outside data starts with a buffer full of entropy
  if (!DBRG_GetEntropy(sizeof(tmpBuf), (BYTE *)&tmpBuf))
    return TPM_RC_NO_RESULT;

  DBRG_Reseed(&drbgDefault, &tmpBuf,
              DbfBuffer(&dfResult, additionalDataSize, additionalData));

  drbgDefault.reseedCounter = 1;

  return TPM_RC_SUCCESS;
}
401  #else
402       // If doing debug, use the input data as the initial setting for the RNG state
403       // so that the test can be reset at any time.
404       // Note: If this is called with a data size of 0 or less, nothing happens. The
405       // presumption is that, in a debug environment, the caller will have specific
406       // values for initialization, so this check is just a simple way to prevent
407       // inadvertent programming errors from screwing things up. This doesn't use an
408       // pAssert() because the non-debug version of this function will accept these
409       // parameters as meaning that there is no additionalData and only hardware
410       // entropy is used.
411       if((additionalDataSize > 0) && (additionalData != NULL))
412           {
413               memset(drbgDefault.seed.bytes, 0, sizeof(drbgDefault.seed.bytes));
414               memcpy(drbgDefault.seed.bytes, additionalData,
415                       MIN(additionalDataSize, sizeof(drbgDefault.seed.bytes)));
416           }
417       drbgDefault.reseedCounter = 1;
418       return TPM_RC_SUCCESS;
419   #endif
420
10.2.17.3.3 CryptRandomGenerate()

Generate a randomSize number or random bytes.

423   LIB_EXPORT UINT16
424   CryptRandomGenerate(
425       INT32            randomSize,
426       BYTE            *buffer
427   )
428   {
429       if(randomSize > UINT16_MAX)
430           randomSize = UINT16_MAX;
431       return DRBG_Generate((RAND_STATE *)&drbgDefault, buffer, (UINT16)randomSize);
432   }
10.2.17.3.3.1 DRBG_InstantiateSeededKdf()

This function is used to instantiate a KDF-based RNG. This is used for derivations. This function always returns TRUE.

```c
LIB_EXPORT BOOL DRBG_InstantiateSeededKdf(
    KDF_STATE       *state,                // OUT: buffer to hold the state
    TPM_ALG_ID       hashAlg,              // IN: hash algorithm
    TPM_ALG_ID       kdf,                   // IN: the KDF to use
    TPM2B           *seed,                  // IN: the seed to use
    const TPM2B     *label,                 // IN: a label for the generation process.
    TPM2B           *context,               // IN: the context value
    UINT32           limit                   // IN: Maximum number of bits from the KDF
)
{
    state->magic = KDF_MAGIC;
    state->limit = limit;
    state->seed = seed;
    state->hash = hashAlg;
    state->kdf = kdf;
    state->label = label;
    state->context = context;
    state->digestSize = CryptHashGetDigestSize(hashAlg);
    state->counter = 0;
    state->residual.t.size = 0;
    return TRUE;
}
```
10.2.17.3.3.2 DRBG_AdditionalData()

Function to reseed the DRBG with additional entropy. This is normally called before computing the protection value of a primary key in the Endorsement hierarchy.

```c
LIBEXPORT void
DRBG_ADDITIONALDATA(
    DRBG_STATE *drbgState,  // IN:OUT state to update
    TPM2B *additionalData  // IN: value to incorporate
)
{
    DRBG_SEED dfResult;
    if(drbgState->magic == DRBG_MAGIC)
    {
        DfBuffer(&dfResult, additionalData->size, additionalData->buffer);
        DRBG_Reseed(drbgState, &dfResult, NULL);
    }
}
```
10.2.17.3.3  DRBG_InstantiateSeeded()

This function is used to instantiate a random number generator from seed values. The nominal use of this generator is to create sequences of pseudo-random numbers from a seed value. This function always returns TRUE.

```
LIB_EXPORT BOOL DRBG_InstantiateSeeded(
    DRBG_STATE *drbgState, // IN/OUT: buffer to hold the state
    const TPM2B *seed, // IN: the seed to use
    const TPM2B *purpose, // IN: a label for the generation process.
    const TPM2B *name, // IN: name of the object
    const TPM2B *additional // IN: additional data
)
{
    DF_STATE         dfState;
    int              totalInputSize;
    // DRBG should have been tested, but...
    if(!IsDrbgTested() && !DRBG_SelfTest())
        FAIL(FATAL_ERROR_SELF_TEST);
    // Initialize the DRBG state
    memset(drbgState, 0, sizeof(DRBG_STATE));
    drbgState->magic = DRBG_MAGIC;

    // Size all of the values
    totalInputSize = (seed != NULL) ? seed->size : 0;
    totalInputSize += (purpose != NULL) ? purpose->size : 0;
    totalInputSize += (name != NULL) ? name->size : 0;
    totalInputSize += (additional != NULL) ? additional->size : 0;

    // Initialize the derivation
    DfStart(&dfState, totalInputSize);

    // Run all the input strings through the derivation function
    if(seed != NULL)
    {
        DfUpdate(&dfState, seed->size, seed->buffer);
    }
    if(purpose != NULL)
    {
        DfUpdate(&dfState, purpose->size, purpose->buffer);
    }
    if(name != NULL)
    {
        DfUpdate(&dfState, name->size, name->buffer);
    }
    if(additional != NULL)
    {
        DfUpdate(&dfState, additional->size, additional->buffer);
    }

    // Used the derivation function output as the "entropy" input. This is not
    // how it is described in SP800-90A but this is the equivalent function
    DRBG_Reseed(((DRBG_STATE *)drbgState), DfEnd(&dfState), NULL);
    return TRUE;
}
```
10.2.17.3.3.4 CryptRandStartup()

This function is called when TPM_Startup() is executed. This function always returns TRUE.

```
512 LIB_EXPORT BOOL
513 CryptRandStartup(
514     void
515 )
516 {
517 #if ! _DRBG_STATE_SAVE
518     // If not saved in NV, re-instantiate on each startup
519     DRBG_Instantiate(&drbgDefault, 0, NULL);
520 #else
521     // If the running state is saved in NV, NV has to be loaded before it can
522     // be updated
523     if(go.drbgState.magic == DRBG_MAGIC)
524         DRBG_Reseed(&go.drbgState, NULL, NULL);
525     else
526         DRBG_Instantiate(&go.drbgState, 0, NULL);
527 #endif
528     return TRUE;
529 }
```
10.2.17.3.3.5 CryptRandInit()

This function is called when _TPM_Init() is being processed.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure</td>
</tr>
</tbody>
</table>

```
LIB_EXPORT BOOL CryptRandInit(
    void
)
{
    #if !USE_DEBUG_RNG
    _plat__GetEntropy(NULL, 0);
    #endif
    return DRBG_SelfTest();
}
```

10.2.17.3.4 DRBG_Generate()

This function generates a random sequence according SP800-90A. If random is not NULL, then randomSize bytes of random values are generated. If random is NULL or randomSize is zero, then the function returns TRUE without generating any bits or updating the reseed counter. This function returns 0 if a reseed is required. Otherwise, it returns the number of bytes produced which could be less than the number requested if the request is too large.

```
LIB_EXPORT UINT16 DRBG_Generate(
    RAND_STATE *state,
    BYTE *random,       // OUT: buffer to receive the random values
    UINT16 randomSize   // IN: the number of bytes to generate
)
{
    if(state == NULL)
        state = (RAND_STATE *)&drbgDefault;

    // If the caller used a KDF state, generate a sequence from the KDF not to exceed the limit.
    if(state->kdf.magic == KDF_MAGIC)
    {
        KDF_STATE *kdf = (KDF_STATE *)state;
        UINT32 counter = (UINT32)kdf->counter;
        INT32 bytesLeft = randomSize;

        if(random == NULL)
            return 0;
        // If the number of bytes to be returned would put the generator over the limit, then return 0
        if(((kdf->counter * kdf->digestSize) + randomSize) * 8) > kdf->limit)
            return 0;
        // Process partial and full blocks until all requested bytes provided
        while(bytesLeft > 0)
        {
            // If there is any residual data in the buffer, copy it to the output buffer
            if(kdf->residual.t.size > 0)
            {
                INT32 size;
```
// Don't use more of the residual than will fit or more than are available
size = MIN(kdf->residual.t.size, bytesLeft);

// Copy some or all of the residual to the output. The residual is at the end of the buffer. The residual might be a full buffer.
MemoryCopy(random,
    &kdf->residual.t.buffer
    [kdf->digestSize - kdf->residual.t.size], size);

    // Advance the buffer pointer
random += size;

    // Reduce the number of bytes left to get
bytesLeft -= size;

    // And reduce the residual size appropriately
kdf->residual.t.size -= (UINT16)size;
}
else
{
    UINT16                     blocks = (UINT16)(bytesLeft / kdf->digestSize);

    // Get the number of required full blocks
    if(blocks > 0)
    {
        UINT16                     size = blocks * kdf->digestSize;

        // Get some number of full blocks and put them in the return buffer
        CryptKDFa(kdf->hash, kdf->seed, kdf->label, kdf->context, NULL,
                  kdf->limit, random, &counter, blocks);

        // reduce the size remaining to be moved and advance the pointer
        bytesLeft -= size;
        random += size;
    }
    else
    {
        // Fill the residual buffer with a full block and then loop to top to get part of it copied to the output.
        kdf->residual.t.size = CryptKDFa(kdf->hash, kdf->seed,
                                         kdf->label, kdf->context, NULL,
                                         kdf->limit,
                                         kdf->residual.t.buffer,
                                         &counter, 1);
    }
    }
    }
    kdf->counter = counter;
    return randomSize;
}
else if(state->drbg.magic == DRBG_MAGIC)
{
    DBRG_STATE                 *drbgState = (DBRG_STATE *)state;
    DBRG_KEY_SCHEDULE          keySchedule;
    DBRG_SEED                  *seed = &drbgState->seed;

    if(drbgState->reseedCounter >= CTR_DRBG_MAX_REQUESTS_PER_RESEED)
    {
        if(drbgState == &drbgDefault)
        {
            DBRG_Reseed(drbgState, NULL, NULL);
            if(IsEntropyBad() && !IsSelfTest())
                return 0;
        }else
        {
            // If this is a PRNG then the only way to get
// here is if the SW has run away.
FAIL(FATAL_ERROR_INTERNAL);
}
// if the allowed number of bytes in a request is larger than the
// less than the number of bytes that can be requested, then check
#if UINT16_MAX >= CTR_DRBG_MAX_BYTES_PER_REQUEST
if(randomSize > CTR_DRBG_MAX_BYTES_PER_REQUEST)
    randomSize = CTR_DRBG_MAX_BYTES_PER_REQUEST;
#endif

// Create encryption schedule
#if (randomSize > CTR_DRBG_MAX_BYTES_PER_REQUEST)
    randomSize = CTR_DRBG_MAX_BYTES_PER_REQUEST;
#endif

// Generate the random data
EncryptDRBG(random, randomSize, &keySchedule, pDRBG_IV(seed),
            drbgState->lastValue);

// Do a key update
DRBG_Update(drbgState, &keySchedule, NULL);

// Increment the reseed counter
    drbgState->reseedCounter += 1;
else
    FAIL(FATAL_ERROR_INTERNAL);
}

return randomSize;

10.2.17.3.5  DRBG_Instantiate()

This is CTR_DRBG_Instantiate_algorithm() from [SP 800-90A 10.2.1.3.1]. This is called when a the TPM DRBG is to be instantiated. This is called to instantiate a DRBG used by the TPM for normal operations.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>instantiation succeeded</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>instantiation failed</td>
</tr>
</tbody>
</table>

LIB_EXPORT BOOL
DRBG_Instantiate(
    DRBG_STATE      *drbgState,
    // OUT: the instantiated value
    UINT16           pSize,
    // IN: Size of personalization string
    BYTE            *personalization
                         // IN: The personalization string
)
{
    DRBG_SEED        seed;
    DRBG_SEED        dfResult;

    // pAssert((pSize == 0) || (pSize <= sizeof(seed)) || (personalization != NULL));
    // If the DRBG has not been tested, test when doing an instantiation. Since
    // Instantiation is called during self test, make sure we don't get stuck in a
    // loop.
    if(!IsDrbgTested() && !IsSelfTest() && !DRBG_SelfTest())
        return FALSE;
    // If doing a self test, DRBG_GetEntropy will return the NIST
    // test vector value.
    if(!DRBG_GetEntropy(sizeof(seed), (BYTE *)&seed))
        return FALSE;
    // set everything to zero
    memset(drbgState, 0, sizeof(DRBG_STATE));
    drbgState->magic = DRBG_MAGIC;
691 // Steps 1, 2, 3, 6, 7 of SP 800-90A 10.2.1.3.1 are exactly what
692 // reseeding does. So, do a reduction on the personalization value (if any)
693 // and do a reseed.
694 DRBG_Reseed(drbgState, &seed, DfBuffer(&dfResult, pSize, personalization));
695
696 return TRUE;
697 }

10.2.17.3.6 DRBG_Uninstantiate()

This is Uninstantiate_function() from [SP 800-90A 9.4].

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>not a valid</td>
</tr>
</tbody>
</table>

698 LIB_EXPORT TPM_RC
699 DRBG_Uninstantiate(
700 DRBG_STATE *drbgState // IN/OUT: working state to erase
701 )
702 {
703 if((drbgState == NULL) || (drbgState->magic != DRBG_MAGIC))
704 return TPM_RC_VALUE;
705 memset(drbgState, 0, sizeof(DRBG_STATE));
706 return TPM_RC_SUCCESS;
707 }
10.2.18 CryptRsa.c

10.2.18.1 Introduction

This file contains implementation of cryptographic primitives for RSA. Vendors may replace the implementation in this file with their own library functions.

10.2.18.2 Includes

Need this define to get the private defines for this function

```c
#define CRYPT_RSA_C
#include "Tpm.h"
#if ALG_RSA
```

10.2.18.3 Obligatory Initialization Functions

10.2.18.3.1 CryptRsaInit()

Function called at _TPM_Init().

```c
BOOL CryptRsaInit(
    void
) {
    return TRUE;
}
```

10.2.18.3.2 CryptRsaStartup()

Function called at TPM2_Startup()

```c
BOOL CryptRsaStartup(
    void
) {
    return TRUE;
}
```

10.2.18.4 Internal Functions

```c
void RsaInitializeExponent(
    privateExponent_t *pExp
) {
    #if CRT_FORMAT_RSA == NO
        BN_INIT(pExp->D);
    #else
        BN_INIT(pExp->Q);
        BN_INIT(pExp->dP);
        BN_INIT(pExp->dQ);
        BN_INIT(pExp->qInv);
    #endif
}
```
10.2.18.4.1 ComputePrivateExponent()

This function computes the private exponent from the primes.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure</td>
</tr>
</tbody>
</table>

```c
32 static BOOL
33 ComputePrivateExponent(
34    bigNum P,  // IN: first prime (size is 1/2 of bnN)
35    bigNum Q,  // IN: second prime (size is 1/2 of bnN)
36    bigNum E,  // IN: the public exponent
37    bigNum N,  // IN: the public modulus
38    privateExponent_t *pExp  // OUT:
39 )
40 {
41    BOOL pOK;
42    BOOL qOK;
43 #if CRT_FORMAT_RSA == NO
44    BN_RSA(bnPhi);
45    // Get compute Phi = (p - 1)(q - 1) = pq - p - q + 1 = n - p - q + 1
46    pOK = BnCopy(bnPhi, N);
47    pOK = pOK && BnSub(bnPhi, bnPhi, P);
48    pOK = pOK && BnSub(bnPhi, bnPhi, Q);
49    pOK = pOK && BnAddWord(bnPhi, bnPhi, 1);
50    // Compute the multiplicative inverse d = 1/e mod Phi
51    pOK = pOK && BnModInverse((bigNum)&pExp->D, E, bnPhi);
52    qOK = pOK;
53 #else
54    BN_PRIME(temp);
55    bigNum pT;
56    // make p the larger value so that m2 is always less than p
57    if(BnUnsignedCmp(P, Q) < 0)
58    {
59        pT = P;
60        P = Q;
61        Q = pT;
62    }
63    //dP = (1/e) mod (p-1) = d mod (p-1)
64    pOK = BnSubWord(temp, P, 1);
65    pOK = pOK && BnModInverse((bigNum)&pExp->dP, E, temp);
66    //dQ = (1/e) mod (q-1) = d mod (q-1)
67    qOK = BnSubWord(temp, Q, 1);
68    qOK = qOK && BnModInverse((bigNum)&pExp->dQ, E, temp);
69    // qInv = (1/q) mod p
70    if(pOK && qOK)
71    {
72        pOK = qOK = BnModInverse((bigNum)&pExp->qInv, Q, P);
73        if(!pOK)
74            BnSetWord(P, 0);
75        if(!qOK)
76            BnSetWord(Q, 0);
77        return pOK && qOK;
78    #endif
79    if(!pOK)
80        BnSetWord(P, 0);
81    if(!qOK)
82        BnSetWord(Q, 0);
83    return pOK && qOK;
84 ```
10.2.18.4.2 RsaPrivateKeyOp()

This function is called to do the exponentiation with the private key. Compile options allow use of the simple (but slow) private exponent, or the more complex but faster CRT method.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure</td>
</tr>
</tbody>
</table>

88 static BOOL
89 RsaPrivateKeyOp(
90     bigNum  inout, // IN/OUT: number to be exponentiated
91     bigNum  N,   // IN: public modulus (can be NULL if CRT)
92     bigNum  p,   // IN: one of the primes (can be NULL if not CRT)
93     privateExponent_t  *pExp
94 )
95 {
96     BOOL     OK;
97     #if CRT_FORMAT_RSA == NO
98         (P);
99     OK = BnModExp(inOut, inOut, (bigNum)&pExp->D, N);
100    #else
101         BN_RSA(M1);
102         BN_RSA(M2);
103         BN_RSA(M);
104         BN_RSA(H);
105         bigNum  Q = (bigNum)&pExp->Q;
106         NOT_REFERENCED(N);
107         // Make P the larger prime.
108         // NOTE that when the CRT form of the private key is created, dP will always
109         // be computed using the larger of p and q so the only thing needed here is that
110         // the primes be selected so that they agree with dP.
111         if(BnUnsignedCmp(P, Q) < 0)
112             {
113                 bigNum  T = P;
114                 P = Q;
115                 Q = T;
116             }
117         // m1 = cdP mod p
118         OK = BnModExp(M1, inout, (bigNum)&pExp->dP, P);
119         // m2 = cdQ mod q
120         OK = OK && BnModExp(M2, inout, (bigNum)&pExp->dQ, Q);
121         // h = qInv * (m1 - m2) mod p = qInv * (m1 + P - m2) mod P because Q < P
122         // so m2 < P
123         OK = OK && BnSub(H, P, M2);
124         OK = OK && BnAdd(H, H, M1);
125         OK = OK && BnModMult(H, H, (bigNum)&pExp->qInv, P);
126         // m = m2 + h * q
127         OK = OK && BnMult(M, H, Q);
128         OK = OK && BnAdd(inOut, M2, M);
129    #endif
130    return OK;
131 }

10.2.18.4.3 RSAEP()

This function performs the RSAEP operation defined in PKCS#1v2.1. It is an exponentiation of a value \( m \) with the public exponent \( e \), modulo the public \( n \).
### Error Returns

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>number to exponentiate is larger than the modulus</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
RSAEP(
    TPM2B  *dInOut,  // IN: size of the encrypted block and the size of
    // the encrypted value. It must be the size of
    // the modulus.
    // OUT: the encrypted data. Will receive the
    // decrypted value
    OBJECT  *key     // IN: the key to use
)
{  
    TPM2B_TYPE(4BYTES, 4);
    TPM2B_4BYTES(e) = {{4, {(BYTE)((RSA_DEFAULT_PUBLIC_EXPONENT >> 24) & 0xff),
    (BYTE)((RSA_DEFAULT_PUBLIC_EXPONENT >> 16) & 0xff),
    (BYTE)((RSA_DEFAULT_PUBLIC_EXPONENT >> 8) & 0xff),
    (BYTE)((RSA_DEFAULT_PUBLIC_EXPONENT) & 0xff)}});
    //
    if(key->publicArea.parameters.rsaDetail.exponent != 0)
        UINT32_TO_BYTE_ARRAY(key->publicArea.parameters.rsaDetail.exponent,
        e.t.buffer);
    return ModExpB(dInOut->size, dInOut->buffer, dInOut->size, dInOut->buffer,
        e.t.size, e.t.buffer, key->publicArea.unique.rsa.t.size,
        key->publicArea.unique.rsa.t.buffer);
}

10.2.18.4.4  RSADP()

This function performs the RSADP operation defined in PKCS#1v2.1. It is an exponentiation of a value \(c\)
with the private exponent \(d\), modulo the public modulus \(n\). The decryption is in place.

This function also checks the size of the private key. If the size indicates that only a prime value is present, the key is converted to being a private exponent.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIZE</td>
<td>the value to decrypt is larger than the modulus</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
RSAADP(
    TPM2B  *inOut,  // IN/OUT: the value to encrypt
    OBJECT  *key     // IN: the key
)
{  
    BN_RSA_INITIALIZED(bnM, inOut);
    BN_RSA_INITIALIZED(bnN, &key->publicArea.unique.rsa);
    BN_RSA_INITIALIZED(bnP, &key->sensitive.sensitive.rsa);
    if(BnUnsignedCmp(bnM, bnN) >= 0)
        return TPM_RC_SIZE;
    // private key operation requires that private exponent be loaded
    // During self-test, this might not be the case so load it up if it hasn’t
    // already done
    // been done
    if(!key->attributes.privateExp)
        CryptRsaloadPrivateExponent(key);
    if(!RsaPrivateKeyOp(bnM, bnN, bnP, &key->privateExponent))
        FAIL(FATAL_ERROR_INTERNAL);
    BnTo2B(bnM, inOut, inOut->size);
    return TPM_RC_SUCCESS;
```
10.2.18.4.5 OaepEncode()

This function performs OAEP padding. The size of the buffer to receive the OAEP padded data must equal the size of the modulus.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>hashAlg is not valid or message size is too large</td>
</tr>
</tbody>
</table>

```c
static TPM_RC OaepEncode(
    TPM2B *padded,  // OUT: the pad data
    TPM_ALG_ID hashAlg,  // IN: algorithm to use for padding
    const TPM2B *label,  // IN: null-terminated string (may be NULL)
    TPM2B *message,  // IN: the message being padded
    RAND_STATE *rand  // IN: the random number generator to use
)
{
    INT32         padLen;
    INT32         dbSize;
    INT32         i;
    BYTE         mySeed[MAX_DIGEST_SIZE];
    BYTE        *seed = mySeed;
    INT32         hLen = CryptHashGetDigestSize(hashAlg);
    BYTE         mask[MAX_RSA_KEY_BYTES];
    BYTE        *pp;
    BYTE        *pm;
    TPM_RC       retVal = TPM_RC_SUCCESS;

    pAssert(padded != NULL && message != NULL);
    // A value of zero is not allowed because the KDF can't produce a result
    // if the digest size is zero.
    if (hLen == 0)
        return TPM_RC_VALUE;

    // Basic size checks
    // make sure digest isn't too big for key size
    if (padded->size < (2 * hLen) + 2)
        ERROR_RETURN(TPM_RC_HASH);

    // and that message will fit messageSize <= k - 2hLen - 2
    if (message->size > (padded->size - (2 * hLen) - 2))
        ERROR_RETURN(TPM_RC_VALUE);

    // Hash L even if it is null
    // Offset into padded leaving room for masked seed and byte of zero
    pp = &padded->buffer[hLen + 1];
    if (CryptHashBlock(hashAlg, label->size, (BYTE *)label->buffer,
                       hLen, pp) != hLen)
        ERROR_RETURN(TPM_RC_FAILURE);

    // concatenate PS of k mLen 2hLen 2
    padLen = padded->size - message->size - (2 * hLen) - 2;
    MemorySet(&pp[hLen], 0, padLen);
    pp[hLen + padLen] = 0x01;
    padLen += 1;
    memcpy(&pp[hLen + padLen], message->buffer, message->size);

    // The total size of db = hLen + pad + mSize;
    dbSize = hLen + padLen + message->size;
```
// If testing, then use the provided seed. Otherwise, use values
// from the RNG
CryptRandomGenerate(hLen, mySeed);
DBG_Generate(rand, mySeed, (UINT16)hLen);
// mask = MGFL (seed, nSize hLen 1)
CryptMGFL(dbSize, mask, hashAlg, hLen, seed);

// Create the masked db
pm = mask;
for (i = dbSize; i > 0; i--)
  *pp++ ^= *pm++;
pp = &padded->buffer[hLen + 1];

// Run the masked data through MGFL
if (CryptMGFL(hLen, &padded->buffer[1], hashAlg, dbSize, pp) != (unsigned)hLen)
  ERROR_RETURN(TPM_RC_VALUE);

// Now XOR the seed to create masked seed
pp = &padded->buffer[1];
pm = seed;
for (i = hLen; i > 0; i--)
  *pp++ ^= *pm++;

// Set the first byte to zero
padded->buffer[0] = 0x00;
Exit:
  return retVal;
}

10.2.18.4.6 OaepDecode()

This function performs OAEP padding checking. The size of the buffer to receive the recovered data. If the padding is not valid, the dSize size is set to zero and the function returns TPM_RC_VALUE.

The dSize parameter is used as an input to indicate the size available in the buffer. If insufficient space is available, the size is not changed and the return code is TPM_RC_VALUE.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>the value to decode was larger than the modulus, or the padding is wrong or the buffer to receive the results is too small</td>
</tr>
</tbody>
</table>

static TPM_RC
OaepDecode(
  TPM2B *dataOut,       // OUT: the recovered data
  TPM_ALG_ID hashAlg,   // IN: algorithm to use for padding
  const TPM2B *label,   // IN: null-terminated string (may be NULL)
  TPM2B *padded        // IN: the padded data
)
{
  UINT32 i;
  BYTE seedMask[MAX_DIGEST_SIZE];
  UINT32 hLen = CryptHashGetDigestSize(hashAlg);
  BYTE mask[MAX_RSA_KEY_BYTES];
  BYTE *pp;
  BYTE *pm;
  TPM_RC retVal = TPM_RC_SUCCESS;

  // Strange size (anything smaller can't be an OAEP padded block)
  // Also check for no leading 0
  if ((padded->size < (unsigned)((2 * hLen) + 2)) || (padded->buffer[0] != 0))
    ERROR_RETURN(TPM_RC_VALUE);

  // Use the hash size to determine what to put through MGFL in order
  // to recover the seedMask
  CryptMGFL(hLen, seedMask, hashAlg, padded->size - hLen - 1,
282  &padded->buffer[hLen + 1]);
283
284  // Recover the seed into seedMask
285  pAssert(hLen <= sizeof(seedMask));
286  pp = &padded->buffer[1];
287  pm = seedMask;
288  for (i = hLen; i > 0; i--)
289      *pm++ ^= *pp++;
290
291  // Use the seed to generate the data mask
292  CryptMGF1(padded->size - hLen - 1, mask, hashAlg, hLen, seedMask);
293
294  // Use the mask generated from seed to recover the padded data
295  pp = &padded->buffer[hLen + 1];
296  pm = mask;
297  for (i = (padded->size - hLen - 1); i > 0; i--)
298      *pm++ ^= *pp++;
299
300  // Make sure that the recovered data has the hash of the label
301  // Put trial value in the seed mask
302  if (!(CryptHashBlock(hashAlg, label->size, (BYTE *)label->buffer, hLen, seedMask)) == hLen)
303      FAIL(FATAL_ERROR_INTERNAL);
304  if (memcmp(seedMask, mask, hLen) != 0)
305      ERROR_RETURN(TPM_RC_VALUE);
306
307  // find the start of the data
308  pm = &mask[hLen];
309  for (i = (UINT32)padded->size - (2 * hLen) - 1; i > 0; i--)
310  {
311      if (*pm++ != 0)
312          break;
313  }
314  // If we ran out of data or didn't end with 0x01, then return an error
315  if (i == 0 || pm[1] != 0x01)
316      ERROR_RETURN(TPM_RC_VALUE);
317
318  // pm should be pointing at the first part of the data
319  // and i is one greater than the number of bytes to move
320  i--;
321  if (i > dataOut->size)
322  {
323      // Special exit to preserve the size of the output buffer
324      return TPM_RC_VALUE;
325      memcpy(dataOut->buffer, pm, i);
326      dataOut->size = (UINT16)i;
327      Exit:
328      if (retVal != TPM_RC_SUCCESS)
329          dataOut->size = 0;
330      return retVal;
331  }

10.2.18.4.7 PKCS1v1_5Encode()

This function performs the encoding for RSAES-PKCS1-V1_5-ENCRYPT as defined in PKCS#1V2.1

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>message size is too large</td>
</tr>
</tbody>
</table>

332
333 static TPM_RC
334 RSAES_PKCS1v1_5Encode()
335     TPM2B *padded,  // OUT: the pad data
336     TPM2B *message,  // IN: the message being padded
337     RAND_STATE *rand

Family "2.0"   TCG Public Review
Level 00 Revision 01.50   Copyright © TCG 2006-2019
September 18, 2018
10.2.18.4.8 RSAES_Decode()

This function performs the decoding for RSAES-PKCS1-V1_5-ENCRYPT as defined in PKCS#1V2.1

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_FAIL</td>
<td>decoding error or results would no fit into provided buffer</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
RSAES_Decode(
    TPM2B *message, // OUT: the recovered message
    TPM2B *coded   // IN: the encoded message
)
{
    BOOL fail = FALSE;
    UINT16 pSize;
    fail = (coded->size < 11);
    fail = (coded->buffer[0] != 0x00) | fail;
    fail = (coded->buffer[1] != 0x02) | fail;
    for(pSize = 2; pSize < coded->size; pSize++)
    {
        if(coded->buffer[pSize] == 0)
            break;
    }
    pSize++;
    // Make sure that pSize has not gone over the end and that there are at least 8
    // bytes of pad data.
    fail = (pSize >= coded->size) | fail;
    fail = ((pSize - 2) < 8) | fail;
    if((message->size < (UINT16)(coded->size - pSize)) || fail)
        return TPM_RC_VALUE;
```
10.2.18.4.9 PssEncode()

This function creates an encoded block of data that is the size of modulus. The function uses the maximum salt size that will fit in the encoded block.

Returns TPM_RC_SUCCESS or goes into failure mode.

```c
static TPM_RC
PssEncode(
    TPM2B *out,  // OUT: the encoded buffer
    TPM_ALG_ID hashAlg, // IN: hash algorithm for the encoding
    TPM2B *digest, // IN: the digest
    RAND_STATE *rand    // IN: random number source
) {
    UINT32               hLen = CryptHashGetDigestSize(hashAlg);
    BYTE                 salt[MAX_RSA_KEY_BYTES - 1];
    UINT16               saltSize;
    BYTE                *ps = salt;
    BYTE                *pOut;
    UINT16               mLen;
    HASH_STATE           hashState;

    // These are fatal errors indicating bad TPM firmware
    pAssert(out != NULL && hLen > 0 && digest != NULL);

    // Get the size of the mask
    mLen = (UINT16)(out->size - hLen - 1);
    // Maximum possible salt size is mask length - 1
    saltSize = mLen - 1;

    // Use the maximum salt size allowed by FIPS 186-4
    if(saltSize > hLen)
        saltSize = (UINT16)hLen;

    //using eOut for scratch space
    // Set the first 8 bytes to zero
    pOut = out->buffer;
    memset(pOut, 0, 8);

    // Get set the salt
    DRBG_Generate(rand, salt, saltSize);

    // Create the hash of the pad || input hash || salt
    CryptHashStart(&hashState, hashAlg);
    CryptDigestUpdate(&hashState, 8, pOut);
    CryptDigestUpdate2B(&hashState, digest);
    CryptDigestUpdate(&hashState, saltSize, salt);
    CryptHashEnd(&hashState, hLen, &pOut[out->size - hLen - 1]);

    // Create a mask
    if(CryptMGF1(mLen, pOut, hashAlg, hLen, &pOut[mLen]) != mLen)
        FAIL(FATAL_ERROR_INTERNAL);

    // Since this implementation uses key sizes that are all even multiples of
    // 8, just need to make sure that the most significant bit is CLEAR
    *pOut &= 0x7f;
```

// Before we mess up the pOut value, set the last byte to 0xbc
pOut[out->size - 1] = 0xbc;

// XOR a byte of 0x01 at the position just before where the salt will be XOR'ed
pOut = &pOut[mLen - saltSize - 1];
*pOut++ ^= 0x01;

// XOR the salt data into the buffer
for (; saltSize > 0; saltSize--)
    *pOut++ ^= *ps++;

// and we are done
return TPM_RC_SUCCESS;

10.2.18.4.10 PssDecode()

This function checks that the PSS encoded block was built from the provided digest. If the check is successful, TPM_RC_SUCCESS is returned. Any other value indicates an error.

This implementation of PSS decoding is intended for the reference TPM implementation and is not at all generalized. It is used to check signatures over hashes and assumptions are made about the sizes of values. Those assumptions are enforce by this implementation. This implementation does allow for a variable size salt value to have been used by the creator of the signature.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SCHEME</td>
<td>hashAlg is not a supported hash algorithm</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>decode operation failed</td>
</tr>
</tbody>
</table>

462 static TPM_RC
463 PssDecode(
464    TPM_ALG_ID   hashAlg,  // IN: hash algorithm to use for the encoding
465    TPM2B       *dIn,     // In: the digest to compare
466    TPM2B       *eIn     // IN: the encoded data
467 )
468 {
469    UINT32           hLen = CryptHashGetDigestSize(hashAlg);
470    BYTE             mask[MAX_RSA_KEY_BYTES];
471    BYTE            *pm = mask;
472    BYTE            *pe;
473    BYTE             pad[8] = {0};
474    UINT32           i;
475    UINT32           mLen;
476    BYTE             fail;
477    TPM_RC           retVal = TPM_RC_SUCCESS;
478    HASH_STATE       hashState;
479
480     // These errors are indicative of failures due to programmer error
481     pAssert(dIn != NULL && eIn != NULL);
482     pe = eIn->buffer;
483
484     // check the hash scheme
485     if (hLen == 0)
486         ERROR_RETURN(TPM_RC_SCHEME);
487     // most significant bit must be zero
488     fail = pe[0] & 0x80;
489     // last byte must be 0xbc
490     fail |= pe[eIn->size - 1] ^ 0xbc;
491
492     // Use the hLen bytes at the end of the buffer to generate a mask
// Doesn't start at the end which is a flag byte
mLen = eIn->size - hLen - 1;
CryptMGF1(mLen, mask, hashAlg, hLen, &pe[mLen]);

// Clear the MSO of the mask to make it consistent with the encoding.
mask[0] &= 0x7F;
pAssert(mLen <= sizeof(mask));

// XOR the data into the mask to recover the salt. This sequence
// advances eIn so that it will end up pointing to the seed data
// which is the hash of the signature data
for(i = mLen; i > 0; i--)
  *pm++ ^= *pe++;

// Find the first byte of 0x01 after a string of all 0x00
for(pm = mask, i = mLen; i > 0; i--)
  if(*pm == 0x01)
    break;
else
  fail |= *pm++;

// i should not be zero
fail |= (i == 0);

// if we have failed, will continue using the entire mask as the salt value so
// that the timing attacks will not disclose anything (I don't think that this
// is a problem for TPM applications but, usually, we don't fail so this
// doesn't cost anything).
if(fail)
  {
    i = mLen;
    pm = mask;
  }
else
  {
    pm++;
    i--;
  }

// i contains the salt size and pm points to the salt. Going to use the input
// hash and the seed to recreate the hash in the lower portion of eIn.
CryptHashStart(&hashState, hashAlg);

// add the pad of 8 zeros
CryptDigestUpdate(&hashState, 8, pad);

// add the provided digest value
CryptDigestUpdate(&hashState, dIn->size, dIn->buffer);

// and the salt
CryptDigestUpdate(&hashState, i, pm);

// get the result
fail |= (CryptHashEnd(&hashState, hLen, mask) != hLen);

// Compare all bytes
for(pm = mask; hLen > 0; hLen--)
  // don't use fail = because that could skip the increment and compare
  // operations after the first failure and that gives away timing
  // information.
  fail |= *pm++ ^ *pe++;

retVal = (fail != 0) ? TPM_RC_VALUE : TPM_RC_SUCCESS;
Exit:
  return retVal;
}
### 10.2.18.4.11 RSASSA_Encode()

Encode a message using PKCS1v1.5 method.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SCHEME</td>
<td>hashAlg is not a supported hash algorithm</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>eOutSize is not large enough</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>hInSize does not match the digest size of hashAlg</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
RSASSA_Encode(
    TPM2B *pOut, // IN:OUT on in, the size of the public key
    TPM_ALG_ID hashAlg, // IN: hash algorithm for PKCS1v1_5
    TPM2B *hIn // IN: digest value to encode)
{
    const BYTE *der;
    BYTE *eOut;
    INT32 derSize = CryptHashGetDer(hashAlg, &der);
    INT32 fillSize;
    TPM_RC retVal = TPM_RC_SUCCESS;

    // Can't use this scheme if the algorithm doesn't have a DER string defined.
    if(derSize == 0)
        ERROR_RETURN(TPM_RC_SCHEME);

    // If the digest size of 'hashAlg' doesn't match the input digest size, then
    // the DER will misidentify the digest so return an error
    if(CryptHashGetDigestSize(hashAlg) != hIn->size)
        ERROR_RETURN(TPM_RC_VALUE);

    fillSize = pOut->size - derSize - hIn->size - 3;
    eOut = pOut->buffer;

    // Make sure that this combination will fit in the provided space
    if(fillSize < 8)
        ERROR_RETURN(TPM_RC_SIZE);

    // Start filling
    *eOut++ = 0; // initial byte of zero
    *eOut++ = 1; // byte of 0x01
    for(; fillSize > 0; fillSize--)
        *eOut++ = 0xff; // bunch of 0xff
    *eOut++ = 0; // another 0
    for(; derSize > 0; derSize--)
        *eOut++ = *der++; // copy the DER
    der = hIn->buffer;
    for(fillSize = hIn->size; fillSize > 0; fillSize--)
        *eOut++ = *der++; // copy the hash

    Exit:
    return retVal;
}
```

### 10.2.18.4.12 RSASSA_Decode()

This function performs the RSASSA decoding of a signature.
### Error Returns

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>decode unsuccessful</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>hashAlg is not supported</td>
</tr>
</tbody>
</table>

```c
static TPM_RC
RSASSA_Decode(
    TPM_ALG_ID       hashAlg, // IN: hash algorithm to use for the encoding
    TPM2B           *hIn,    // In: the digest to compare
    TPM2B           *eIn     // IN: the encoded data
) {
    BYTE             fail;
    const BYTE      *der;
    BYTE            *pe;
    INT32            derSize = CryptHashGetDer(hashAlg, &der);
    INT32            hashSize = CryptHashGetDigestSize(hashAlg);
    INT32            fillSize;
    TPM_RC           retVal;
    BYTE             *digest;
    UINT16           digestSize;

    pAssert(hIn != NULL && eIn != NULL);
    pe = eIn->buffer;

    // Can't use this scheme if the algorithm doesn't have a DER string
    // defined or if the provided hash isn't the right size
    if(derSize == 0 || ((unsigned)hashSize != hIn->size))
        ERROR_RETURN(TPM_RC_SCHEME);

    // Make sure that this combination will fit in the provided space
    // Since no data movement takes place, can just walk though this
    // and accept nearly random values. This can only be called from
    // CryptValidateSignature() so eInSize is known to be in range.
    fillSize = eIn->size - derSize - hashSize - 3;

    // Start checking (fail will become non-zero if any of the bytes do not have
    // the expected value.
    fail = *pe++;    // initial byte of zero
    fail |= *pe++ ^ 1; // byte of 0x01
    for(; fillSize > 0; fillSize--)
        fail |= *pe++ ^ 0xff; // bunch of 0xff
    fail |= *pe++;    // another 0
    for(; derSize > 0; derSize--)
        fail |= *pe++ ^ *der++; // match the DER
    digestSize = hIn->size;
    digest = hIn->buffer;
    for(; digestSize > 0; digestSize--)
        fail |= *pe++ ^ *digest++; // match the hash
    retVal = (fail != 0) ? TPM_RC_VALUE : TPM_RC_SUCCESS;

    Exit:
    return retVal;
}
```

### 10.2.18.5 Externally Accessible Functions

#### 10.2.18.5.1 CryptRsaSelectScheme()

This function is used by TPM2_RSA_Decrypt() and TPM2_RSA_Encrypt(). It sets up the rules to select a scheme between input and object default. This function assume the RSA object is loaded. If a default scheme is defined in object, the default scheme should be chosen, otherwise, the input scheme should be chosen. In the case that both the object and scheme are not TPM_ALG_NULL, then if the schemes
are the same, the input scheme will be chosen. If the scheme are not compatible, a NULL pointer will be returned.

The return pointer may point to a TPM_ALG_NULL scheme.

```c
TPMT_RSA_DECRYPT* CryptRsaSelectScheme(
    TPMI_DH_OBJECT rsaHandle,  // IN: handle of an RSA key
    TPMT_RSA_DECRYPT *scheme    // IN: a sign or decrypt scheme
)
{
    OBJECT *rsaObject;
    TPMT_ASYM_SCHEME *keyScheme;
    TPMT_RSA_DECRYPT *retVal = NULL;

    // Get sign object pointer
    rsaObject = HandleToObject(rsaHandle);
    keyScheme = &rsaObject->publicArea.parameters.asymDetail.scheme;

    // if the default scheme of the object is TPM_ALG_NULL, then select the
    // input scheme
    if(keyScheme->scheme == TPM_ALG_NULL)
    {
        retVal = scheme;
    }
    // if the object scheme is not TPM_ALG_NULL and the input scheme is
    // ALG_NULL, then select the default scheme of the object.
    else if(scheme->scheme == TPM_ALG_NULL)
    {
        // if input scheme is NULL
        retVal = (TPMT_RSA_DECRYPT *)keyScheme;
    }
    // get here if both the object scheme and the input scheme are
    // not TPM ALG NULL. Need to insure that they are the same.
    // IMPLEMENTATION NOTE: This could cause problems if future versions have
    // schemes that have more values than just a hash algorithm. A new function
    // (IsSchemeSame()) might be needed then.
    else if((keyScheme->scheme != scheme->scheme)
            && (keyScheme->details.anySig.hashAlg != scheme->details.anySig.hashAlg))
    {
        retVal = scheme;
    }

    // two different, incompatible schemes specified will return NULL
    return retVal;
}
```

### 10.2.18.5.2 CryptRsaLoadPrivateExponent()

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_BINDING</td>
<td>public and private parts of rsaKey are not matched</td>
</tr>
</tbody>
</table>

```c
TPM_RC CryptRsaLoadPrivateExponent(
    OBJECT *rsaKey  // IN: the RSA key object
)
{
    BN_RSA_INITIALIZED(bnN, &rsaKey->publicArea.unique.rsa);
    BN_PRIME_INITIALIZED(bnP, &rsaKey->sensitive.sensitive.rsa);
    BN_RSA(bnQ);
    BN_PRIME(bnQr);
    BN_WORD_INITIALIZED(bnE, (rsaKey->publicArea.parameters.rsaDetail.exponent == 0) ? RSA_DEFAULT_PUBLIC_EXPONENT
```
703 : rsaKey->publicArea.parameters.rsaDetail.exponent);
704 TPM_RC
705     retVal = TPM_RC_SUCCESS;
706   if(!rsaKey->attributes.privateExp)
707     { TEST(ALG_NULL_VALUE);
708       // Make sure that the bigNum used for the exponent is properly initialized
709       RsaInitializeExponent(&rsaKey->privateExponent);
710       // Find the second prime by division
711       BnDiv(bnQ, bnQr, bnN, bnP);
712       if(!BnEqualZero(bnQr))
713         ERROR_RETURN(TPM_RC_BINDING);
714       // Compute the private exponent and return it if found
715       if(!ComputePrivateExponent(bnP, bnQ, bnE, bnN,
716         &rsaKey->privateExponent))
717         ERROR_RETURN(TPM_RC_BINDING);
718   }
719   Exit:
720   rsaKey->attributes.privateExp = (retVal == TPM_RC_SUCCESS);
721     return retVal;
722 }
723
724 }
725
10.2.18.5.3 CryptRsaEncrypt()

This is the entry point for encryption using RSA. Encryption is use of the public exponent. The padding parameter determines what padding will be used.

The cOutSize parameter must be at least as large as the size of the key.

If the padding is RSA_PAD_NONE, dIn is treated as a number. It must be lower in value than the key modulus.

NOTE: If dIn has fewer bytes than cOut, then we don’t add low-order zeros to dIn to make it the size of the RSA key for the call to RSAEP. This is because the high order bytes of dIn might have a numeric value that is greater than the value of the key modulus. If this had low-order zeros added, it would have a numeric value larger than the modulus even though it started out with a lower numeric value.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>cOutSize is too small (must be the size of the modulus)</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>padType is not a supported scheme</td>
</tr>
</tbody>
</table>

LIB_EXPORT TPM_RC

CryptRsaEncrypt()

727 TPM2B_PUBLIC_KEY_RSA *cOut,  // OUT: the encrypted data
728 TPM2B *dIn,      // IN: the data to encrypt
729 OBJECT *key,     // IN: the key used for encryption
730 TPM_RSA_DECRYPT *scheme, // IN: the type of padding and hash
731       // if needed
732 const TPM2B *label, // IN: in case it is needed
733 RAND_STATE *rand   // IN: random number generator
734       // state (mostly for testing)
735 }
736
737 TPM_RC
738     retVal = TPM_RC_SUCCESS;
739 TPM2B_PUBLIC_KEY_RSA dataIn;
740   // if the input and output buffers are the same, copy the input to a scratch
741   // buffer so that things don't get messed up.
742   if(dIn == &cOut->b)
743     { MemoryCopy2B(&dataIn.b, dIn, sizeof(dataIn.t.buffer));
745     dIn = &dataIn.b;
746 }
747     // All encryption schemes return the same size of data
748     cOut->t.size = key->publicArea.unique.rsa.t.size;
749     TEST(scheme->scheme);
750     switch (scheme->scheme)
751     {
752       case ALG_NULL_VALUE: // 'raw' encryption
753         {
754           INT32 i;
755           INT32 dSize = dIn->size;
756           // dIn can have more bytes than cOut as long as the extra bytes
757           // are zero. Note: the more significant bytes of a number in a byte
758           // buffer are the bytes at the start of the array.
759           for (i = 0; (i < dSize) && (dIn->buffer[i] == 0); i++);
760           dSize -= i;
761           if (dSize > cOut->t.size)
762             ERROR_RETURN(TPM_RC_VALUE);
763             // Pad cOut with zeros if dIn is smaller
764             memset(cOut->t.buffer, 0, cOut->t.size - dSize);
765             // And copy the rest of the value
766             memcpy(&cOut->t.buffer[cOut->t.size - dSize], &dIn->buffer[i], dSize);
767             // If the size of dIn is the same as cOut dIn could be larger than
768             // the modulus. If it is, then RSAEP() will catch it.
769         }
770         break;
771        case ALG_RSAES_VALUE:
772             retVal = RSAES_PKCS1v1_5Encode(&cOut->b, dIn, rand);
773             break;
774        case ALG_OAEP_VALUE:
775             retVal = OaepEncode(&cOut->b, scheme->details.oaep.hashAlg, label, dIn,
776                     rand);
777             break;
778        default:
779             ERROR_RETURN(TPM_RC_SCHEME);
780             break;
781       }
782       // All the schemes that do padding will come here for the encryption step
783       // Check that the Encoding worked
784       if (retVal == TPM_RC_SUCCESS)
785         // Padding OK so do the encryption
786         retVal = RSAEP(&cOut->b, key);
787     Exit:
788         return retVal;
789     }
790  }
791
792 10.2.18.5.4 CryptRsaDecrypt()
793
794 This is the entry point for decryption using RSA. Decryption is use of the private exponent. The padType
795 parameter determines what padding was used.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIZE</td>
<td>cInSize is not the same as the size of the public modulus of key; or</td>
</tr>
<tr>
<td></td>
<td>numeric value of the encrypted data is greater than the modulus</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>dOutSize is not large enough for the result</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>padType is not supported</td>
</tr>
</tbody>
</table>

792 LIB_EXPORT TPM_RC
793 CryptRsaDecrypt(
794     TPM2B *dOut,    // OUT: the decrypted data
795 )
795 TPM2B *cIn,      // IN: the data to decrypt
796 OBJECT *key,     // IN: the key to use for decryption
797 TPMT_RSA_DECRYPT *scheme,  // IN: the padding scheme
798 const TPM2B *label // IN: in case it is needed for the scheme
799 }
800 {
801 TPM_RC retVal;
802 // Make sure that the necessary parameters are provided
803 pAssert(cIn != NULL && dOut != NULL && key != NULL);
804 // Size is checked to make sure that the encrypted value is the right size
805 if(cIn->size != key->publicArea.unique.rsa.t.size)
806 ERROR_RETURN(TPM_RC_SIZE);
807 TEST(scheme->scheme);
808 // For others that do padding, do the decryption in place and then
809 // go handle the decoding.
810 if(retVal == TPM_RC_SUCCESS)
811 {
812 // Remove padding
813 switch(scheme->scheme)
814 {
815 case ALG_NULL_VALUE:
816 if(dOut->size < cIn->size)
817     return TPM_RC_VALUE;
818     MemoryCopy2B(dOut, cIn, dOut->size);
819     break;
820 case ALG_RSAES_VALUE:
821     retVal = RSAES_Decode(dOut, cIn);
822     break;
823 case ALG_OAEP_VALUE:
824     retVal = OaepDecode(dOut, scheme->details.oaep.hashAlg, label, cIn);
825     break;
826 default:
827     retVal = TPM_RC_SCHEMA;
828     break;
829 }
830 Exit:
831 return retVal;
832 }

10.2.18.5.5 CryptRsaSign()

This function is used to generate an RSA signature of the type indicated in `scheme`.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SCHEMA</td>
<td><code>scheme</code> or <code>hashAlg</code> are not supported</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td><code>hInSize</code> does not match <code>hashAlg</code> (for RSASSA)</td>
</tr>
</tbody>
</table>

```c
839 LIB_EXPORT TPM_RC
840 CryptRsaSign(
841    TPMT_SIGNATURE *sigOut,
842    OBJECT *key,     // IN: key to use
843    TPM2B_DIGEST *hIn,  // IN: the digest to sign
844    RAND_STATE *rand  // IN: the random number generator
845    // to use (mostly for testing)
846 )
847 {
```
TPM_RC    retVal = TPM_RC_SUCCESS;
UINT16    modSize;

// parameter checks
pAssert(sigOut != NULL && key != NULL && hIn != NULL);
modSize = key->publicArea.unique.rsa.t.size;

// for all non-null signatures, the size is the size of the key modulus
sigOut->signature.rsapss.sig.t.size = modSize;
TEST(sigOut->sigAlg);

switch(sigOut->sigAlg)
{
    case ALG_NULL_VALUE:
        sigOut->signature.rsapss.sig.t.size = 0;
        return TPM_RC_SUCCESS;
    case ALG_RSAPSS_VALUE:
        retVal = PssEncode(&sigOut->signature.rsapss.sig.b,
            sigOut->signature.rsapss.hash, &hIn->b, rand);
        break;
    case ALG_RSASSA_VALUE:
        retVal = RSASSA_Encode(&sigOut->signature.rsassa.sig.b,
            sigOut->signature.rsassa.hash, &hIn->b);
        break;
    default:
        retVal = TPM_RC_SCHEME;
}
if(retVal == TPM_RC_SUCCESS)
{
    // Do the encryption using the private key
    retVal = RSADP(&sigOut->signature.rsapss.sig.b, key);
}
return retVal;

10.2.18.5.6 CryptRsaValidateSignature()

This function is used to validate an RSA signature. If the signature is valid TPM_RC_SUCCESS is
returned. If the signature is not valid, TPM_RC_SIGNATURE is returned. Other return codes indicate
either parameter problems or fatal errors.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIGNATURE</td>
<td>the signature does not check</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>unsupported scheme or hash algorithm</td>
</tr>
</tbody>
</table>

LIB_EXPORT TPM_RC
CryptRsaValidateSignature(
TPMT_SIGNATURE *sig, // IN: signature
OBJECT      *key,     // IN: public modulus
TPM2B_DIGEST *digest  // IN: The digest being validated
)
{
    TPM_RC    retVal;
    // Fatal programming errors
    pAssert(key != NULL && sig != NULL && digest != NULL);
    switch(sig->sigAlg)
    {
        case ALG_RSAPSS_VALUE:
        case ALG_RSASSA_VALUE:

    }
break;
default:
    return TPM_RC_SCHEME;
}

// Errors that might be caused by calling parameters
if(sig->signature.rsassa.sig.t.size != key->publicArea.unique.rsa.t.size)
    ERROR_RETURN(TPM_RC_SIGNATURE);

TEST(sig->sigAlg);

// Decrypt the block
retVal = RSAEP(&sig->signature.rsassa.sig.b, key);
if(retVal == TPM_RC_SUCCESS)
{
    switch(sig->sigAlg)
    {
    case ALG_RSAPSS_VALUE:
        retVal = PssDecode(sig->signature.any.hashAlg, &digest->b, &sig->signature.rsassa.sig.b);
        break;
    case ALG_RSASSA_VALUE:
        retVal = RSASSA_Decode(sig->signature.any.hashAlg, &digest->b, &sig->signature.rsassa.sig.b);
        break;
    default:
        return TPM_RC_SCHEME;
    }
}
Exit:
    return (retVal != TPM_RC_SUCCESS) ? TPM_RC_SIGNATURE : TPM_RC_SUCCESS;
}

#if SIMULATION && USE_RSA_KEY_CACHE
extern int s_rsaKeyCacheEnabled;
int GetCachedRsaKey(OBJECT *key, RAND_STATE *rand);
#define GET_CACHED_KEY(key, rand) 
        (s_rsaKeyCacheEnabled && GetCachedRsaKey(key, rand))
#else
#define GET_CACHED_KEY(key, rand)
#endif

10.2.18.5.7 CryptRsaGenerateKey()

Generate an RSA key from a provided seed

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CANCELED</td>
<td>operation was canceled</td>
</tr>
<tr>
<td>TPM_RC_RANGE</td>
<td>public exponent is not supported</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>could not find a prime using the provided parameters</td>
</tr>
</tbody>
</table>

LIB_EXPORT TPM_RC
CryptRsaGenerateKey(
OBJECT *rsaKey,       // IN/OUT: The object structure in which
RAND_STATE *rand       // IN: if not NULL, the deterministic
)
{
    UINT32 i;
    BN_PRIME(bnP); // These four declarations initialize the number to 0
    BN_PRIME(bnQ);
    BN_RSA(bnd);
BN_RSA(bnN);
BN_WORD(bnE);
UINT32 e;
int keySizeInBits;
TPMT_PUBLIC *publicArea = rsaKey->publicArea;
TPMT_SENSITIVE *sensitive = rsaKey->sensitive;
TPM_RC retVal = TPM_RC_NO_RESULT;

//
// Need to make sure that the caller did not specify an exponent that is not supported
//
if (e == 0)
    e = RSA_DEFAULT_PUBLIC_EXPONENT;
if (e < 65537)
    ERROR_RETURN(TPM_RC_RANGE);
if (e != RSA_DEFAULT_PUBLIC_EXPONENT && !IsPrimeInt(e))
    ERROR_RETURN(TPM_RC_RANGE);
BnSetWord(bnE, e);
// Check that e is prime
// check for supported key size.
keySizeInBits = rsaKey->parameters.rsaDetail.keyBits;
if (((keySizeInBits % 1024) != 0) || (keySizeInBits > MAX_RSA_KEY_BITS))
    ERROR_RETURN(TPM_RC_VALUE);

// Set the prime size for instrumentation purposes
INSTRUMENT_SET(PrimeIndex, PRIME_INDEX(keySizeInBits / 2));

#if SIMULATION && USE_RSA_KEY_CACHE
if (GET_CACHED_KEY(rsaKey, rand))
    return TPM_RC_SUCCESS;
#endif

// Make sure that key generation has been tested
TEST(ALG_NULL_VALUE);

// Need to initialize the privateExponent structure
RsaInitializeExponent(rsaKey->privateExponent);

// The prime is computed in P. When a new prime is found, Q is checked to see if it is zero. If so, P is copied to Q and a new P is found.
// When both P and Q are non-zero, the modulus and private exponent are computed and a trial encryption/decryption is performed. If the encrypt/decrypt fails, assume that at least one of the primes is composite. Since we don't know which one, set Q to zero and start over and find a new pair of primes.

for(i = 1; (retVal != TPM_RC_SUCCESS) && (i != 100); i++)
{
    if (_plat__IsCanceled())
        ERROR_RETURN(TPM_RC_CANCELED);
    BnGeneratePrimeForRSA(bnP, keySizeInBits / 2, e, rand);
    INSTRUMENT_INC(PrimeCounts[PrimeIndex]);
    // If this is the second prime, make sure that it differs from the first prime by at least 2^100
    if (BnEqualZero(bnQ))
    {
        // copy p to q and compute another prime in p
        BnCopy(bnQ, bnP);
        continue;
    }
    // Make sure that the difference is at least 100 bits. Need to do it this
// way because the big numbers are only positive values
if (BnUnsignedCmp(bnP, bnQ) < 0)
    BnSub(bnD, bnQ, bnP);
else
    BnSub(bnD, bnP, bnQ);
if (BnMsb(bnD) < 100)
    continue;

// Form the public modulus and set the unique value
BnMult(bnN, bnP, bnQ);
BnTo2B(bnN, &publicArea->unique.rsa.b,
        (NUMBYTES)BITS_TO_BYTES(keySizeInBits));

// And the prime to the sensitive area
BnTo2B(bnP, &sensitive->sensitive.rsa.b,
        (NUMBYTES)BITS_TO_BYTES(keySizeInBits) / 2);

// Make sure everything came out right. The MSb of the values must be
// one
if (((publicArea->unique.rsa.t.buffer[0] & 0x80) == 0)
    || ((sensitive->sensitive.rsa.t.buffer[0] & 0x80) == 0))
    FAIL(FATAL_ERROR_INTERNAL);

// Make sure that we can form the private exponent values
if (ComputePrivateExponent(bnP, bnQ, bnE, bnN, &rsaKey->privateExponent) != TRUE)
{
    // If ComputePrivateExponent could not find an inverse for
    // Q, then copy P and recompute P. This might
    // cause both to be recomputed if P is also zero
    if (BnEqualZero(bnQ))
        BnCopy(bnQ, bnP);
    continue;
}
retVal = TPM_RC_SUCCESS;

// Do a trial encryption decryption if this is a signing key
if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
{
    BN_RSA(temp1);
    BN_RSA(temp2);
    BnGenerateRandomInRange(temp1, bnN, rand);

    // Encrypt with public exponent...
    BnModExp(temp2, temp1, bnE, bnN);
    // ... then decrypt with private exponent
    RsaPrivateKeyOp(temp2, bnN, bnP, &rsaKey->privateExponent);

    // If the starting and ending values are not the same,
    // start over...-
    if (BnUnsignedCmp(temp2, temp1) != 0)
    {
        BnSetWord(bnQ, 0);
        retVal = TPM_RC_NO_RESULT;
    }
}

// Make sure that we can form the private exponent values
if (ComputePrivateExponent(bnP, bnQ, bnE, bnN, &rsaKey->privateExponent) != TRUE)
{
    // If ComputePrivateExponent could not find an inverse for
    // Q, then copy P and recompute P. This might
    // cause both to be recomputed if P is also zero
    if (BnEqualZero(bnQ))
        BnCopy(bnQ, bnP);
    continue;
}
retVal = TPM_RC_SUCCESS;

// Do a trial encryption decryption if this is a signing key
if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
{
    BN_RSA(temp1);
    BN_RSA(temp2);
    BnGenerateRandomInRange(temp1, bnN, rand);

    // Encrypt with public exponent...
    BnModExp(temp2, temp1, bnE, bnN);
    // ... then decrypt with private exponent
    RsaPrivateKeyOp(temp2, bnN, bnP, &rsaKey->privateExponent);

    // If the starting and ending values are not the same,
    // start over...-
    if (BnUnsignedCmp(temp2, temp1) != 0)
    {
        BnSetWord(bnQ, 0);
        retVal = TPM_RC_NO_RESULT;
    }
}

// Make sure that we can form the private exponent values
if (ComputePrivateExponent(bnP, bnQ, bnE, bnN, &rsaKey->privateExponent) != TRUE)
{
    // If ComputePrivateExponent could not find an inverse for
    // Q, then copy P and recompute P. This might
    // cause both to be recomputed if P is also zero
    if (BnEqualZero(bnQ))
        BnCopy(bnQ, bnP);
    continue;
}
retVal = TPM_RC_SUCCESS;

// Do a trial encryption decryption if this is a signing key
if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
{
    BN_RSA(temp1);
    BN_RSA(temp2);
    BnGenerateRandomInRange(temp1, bnN, rand);

    // Encrypt with public exponent...
    BnModExp(temp2, temp1, bnE, bnN);
    // ... then decrypt with private exponent
    RsaPrivateKeyOp(temp2, bnN, bnP, &rsaKey->privateExponent);

    // If the starting and ending values are not the same,
    // start over...-
    if (BnUnsignedCmp(temp2, temp1) != 0)
    {
        BnSetWord(bnQ, 0);
        retVal = TPM_RC_NO_RESULT;
    }
}

// Make sure that we can form the private exponent values
if (ComputePrivateExponent(bnP, bnQ, bnE, bnN, &rsaKey->privateExponent) != TRUE)
{
    // If ComputePrivateExponent could not find an inverse for
    // Q, then copy P and recompute P. This might
    // cause both to be recomputed if P is also zero
    if (BnEqualZero(bnQ))
        BnCopy(bnQ, bnP);
    continue;
}
retVal = TPM_RC_SUCCESS;

// Do a trial encryption decryption if this is a signing key
if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
{
    BN_RSA(temp1);
    BN_RSA(temp2);
    BnGenerateRandomInRange(temp1, bnN, rand);

    // Encrypt with public exponent...
    BnModExp(temp2, temp1, bnE, bnN);
    // ... then decrypt with private exponent
    RsaPrivateKeyOp(temp2, bnN, bnP, &rsaKey->privateExponent);

    // If the starting and ending values are not the same,
    // start over...-
    if (BnUnsignedCmp(temp2, temp1) != 0)
    {
        BnSetWord(bnQ, 0);
        retVal = TPM_RC_NO_RESULT;
    }
}

// Make sure that we can form the private exponent values
if (ComputePrivateExponent(bnP, bnQ, bnE, bnN, &rsaKey->privateExponent) != TRUE)
{
    // If ComputePrivateExponent could not find an inverse for
    // Q, then copy P and recompute P. This might
    // cause both to be recomputed if P is also zero
    if (BnEqualZero(bnQ))
        BnCopy(bnQ, bnP);
    continue;
}
retVal = TPM_RC_SUCCESS;

// Do a trial encryption decryption if this is a signing key
if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
{
    BN_RSA(temp1);
    BN_RSA(temp2);
    BnGenerateRandomInRange(temp1, bnN, rand);

    // Encrypt with public exponent...
    BnModExp(temp2, temp1, bnE, bnN);
    // ... then decrypt with private exponent
    RsaPrivateKeyOp(temp2, bnN, bnP, &rsaKey->privateExponent);

    // If the starting and ending values are not the same,
    // start over...-
    if (BnUnsignedCmp(temp2, temp1) != 0)
    {
        BnSetWord(bnQ, 0);
        retVal = TPM_RC_NO_RESULT;
    }
}

// Make sure that we can form the private exponent values
if (ComputePrivateExponent(bnP, bnQ, bnE, bnN, &rsaKey->privateExponent) != TRUE)
{
    // If ComputePrivateExponent could not find an inverse for
    // Q, then copy P and recompute P. This might
    // cause both to be recomputed if P is also zero
    if (BnEqualZero(bnQ))
        BnCopy(bnQ, bnP);
    continue;
}
retVal = TPM_RC_SUCCESS;

// Do a trial encryption decryption if this is a signing key
if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
{
    BN_RSA(temp1);
    BN_RSA(temp2);
    BnGenerateRandomInRange(temp1, bnN, rand);

    // Encrypt with public exponent...
    BnModExp(temp2, temp1, bnE, bnN);
    // ... then decrypt with private exponent
    RsaPrivateKeyOp(temp2, bnN, bnP, &rsaKey->privateExponent);

    // If the starting and ending values are not the same,
    // start over...-
    if (BnUnsignedCmp(temp2, temp1) != 0)
    {
        BnSetWord(bnQ, 0);
        retVal = TPM_RC_NO_RESULT;
    }
}

// Make sure that we can form the private exponent values
if (ComputePrivateExponent(bnP, bnQ, bnE, bnN, &rsaKey->privateExponent) != TRUE)
{
    // If ComputePrivateExponent could not find an inverse for
    // Q, then copy P and recompute P. This might
    // cause both to be recomputed if P is also zero
    if (BnEqualZero(bnQ))
        BnCopy(bnQ, bnP);
    continue;
}
retVal = TPM_RC_SUCCESS;

// Do a trial encryption decryption if this is a signing key
if (IS_ATTRIBUTE(publicArea->objectAttributes, TPMA_OBJECT, sign))
{
    BN_RSA(temp1);
    BN_RSA(temp2);
    BnGenerateRandomInRange(temp1, bnN, rand);

    // Encrypt with public exponent...
    BnModExp(temp2, temp1, bnE, bnN);
    // ... then decrypt with private exponent
    RsaPrivateKeyOp(temp2, bnN, bnP, &rsaKey->privateExponent);

    // If the starting and ending values are not the same,
    // start over...-
    if (BnUnsignedCmp(temp2, temp1) != 0)
    {
        BnSetWord(bnQ, 0);
        retVal = TPM_RC_NO_RESULT;
    }
}
10.2.19 CryptSmac.c

10.2.19.1 Introduction

This file contains the implementation of the message authentication codes based on a symmetric block cipher. These functions only use the single block encryption functions of the selected symmetric cryptographic library.

10.2.19.2 Includes, Defines, and Typedefs

```c
#define _CRYPT_HASH_C_
#include "Tpm.h"
#if SMAC_IMPLEMENTED
```

10.2.19.2.1 CryptSmacStart()

Function to start an SMAC.

```c
UINT16 CryptSmacStart(
    HASH_STATE *state,
    TPMU_PUBLIC_PARMS *keyParameters,
    TPM_ALG_ID macAlg, // IN: the type of MAC
    TPM2B *key
)
```

```c
UINT16 retVal = 0;
// Make sure that the key size is correct. This should have been checked
// at key load, but...
if (BITS_TO_BYTES(keyParameters->symDetail.sym.keyBits.sym) == key->size)
{
    switch (macAlg)
    {
    #if ALG_CMAC
    case ALG_CMAC_VALUE:
        retVal = CryptCmacStart(&state->state.smac, keyParameters, macAlg, key);
        break;
    #endif
    default:
        break;
    }
    state->type = (retVal != 0) ? HASH_STATE_SMAC : HASH_STATE_EMPTY;
    return retVal;
}
```

10.2.19.2.2 CryptMacStart()

Function to start either an HMAC or an SMAC. Cannot reuse the CryptHmacStart() function because of the difference in number of parameters.

```c
UINT16 CryptMacStart(
    HMAC_STATE *state,
    TPMU_PUBLIC_PARMS *keyParameters,
    TPM_ALG_ID macAlg, // IN: the type of MAC
    TPM2B *key
)
```
```c
MemorySet(state, 0, sizeof(HMAC_STATE));
if(CryptHashIsValidAlg(macAlg, FALSE))
    return CryptHmacStart(state, macAlg, key->size, key->buffer);
else if(CryptSmacIsValidAlg(macAlg, FALSE))
    return CryptSmacStart(&state->hashState, keyParameters, macAlg, key);
else
    return 0;
}

10.2.19.2.3 CryptMacEnd()

Dispatch to the MAC end function using a size and buffer pointer.

```
10.2.20 CryptSym.c

10.2.20.1 Introduction

This file contains the implementation of the symmetric block cipher modes allowed for a TPM. These functions only use the single block encryption functions of the selected symmetric crypto library.

10.2.20.2 Includes, Defines, and Typedefs

```c
#include "Tpm.h"
#include "CryptSym.h"
```

10.2.20.3 Initialization and Data Access Functions

10.2.20.3.1 CryptSymInit()

This function is called to do _TPM_Init() processing

```c
BOOL CryptSymInit()
{
    void

    { return TRUE;
}
```

10.2.20.3.2 CryptSymStartup()

This function is called to do TPM2_Startup() processing

```c
BOOL CryptSymStartup()
{
    void

    { return TRUE;
}
```

10.2.20.3.3 CryptGetSymmetricBlockSize()

This function returns the block size of the algorithm.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 0</td>
<td>cipher not supported</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>the cipher block size in bytes</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT INT16 CryptGetSymmetricBlockSize()
{
    TPM_ALG_ID symmetricAlg,  // IN: the symmetric algorithm
    UINT16    keySizeInBits   // IN: the key size

    switch(symmetricAlg)
    {
    #if ALG_AES
```
```c
    case ALG_AES_VALUE:
        switch(keySizeInBits)
        {
            case 128:
                return AES_128_BLOCK_SIZE_BYTES;
            case 192:
                return AES_192_BLOCK_SIZE_BYTES;
            case 256:
                return AES_256_BLOCK_SIZE_BYTES;
            default:
                break;
        }
    break;
#endif
#if ALG_SM4
    case ALG_SM4_VALUE:
        switch(keySizeInBits)
        {
            case 128:
                return SM4_128_BLOCK_SIZE_BYTES;
            default:
                break;
        }
#endif
#if ALG_CAMELLIA
    case ALG_CAMELLIA_VALUE:
        switch(keySizeInBits)
        {
            case 128:
                return CAMELLIA_128_BLOCK_SIZE_BYTES;
            case 192:
                return CAMELLIA_192_BLOCK_SIZE_BYTES;
            case 256:
                return CAMELLIA_256_BLOCK_SIZE_BYTES;
            default:
                break;
        }
#endif
#if ALG_TDES
    case ALG_TDES_VALUE:
        switch(keySizeInBits)
        {
            case 128:
                return TDES_128_BLOCK_SIZE_BYTES;
            case 192:
                return TDES_192_BLOCK_SIZE_BYTES;
            default:
                break;
        }
#endif
    default:
        break;
    return 0;
```
Error Returns | Meaning
--- | ---
TPM_RC_SIZE | dSize is not a multiple of the block size for an algorithm that requires it
TPM_RC_FAILURE | Fatal error

```c
LIB_EXPORT TPM_RC
CryptSymmetricEncrypt(
    BYTE *dOut,                  // OUT:
    TPM_ALG_ID algorithm,       // IN: the symmetric algorithm
    UINT16 keySizeInBits,       // IN: key size in bits
    const BYTE *key,            // IN: key buffer. The size of this buffer
                                // in bytes is (keySizeInBits + 7) / 8
    TPM2B_IV *ivInOut,          // IN/OUT: IV for decryption.
    TPM_ALG_ID mode,            // IN: Mode to use
    INT32 dSize,                // IN: data size (may need to be a
                                // multiple of the blockSize)
    const BYTE *dIn              // IN: data buffer
)
{
    BYTE *pIv;
    int i;
    BYTE tmp[MAX_SYM_BLOCK_SIZE];
    BYTE *pT;
    tpmCryptKeySchedule_t keySchedule;
    INT16 blockSize;
    TpmCryptSetSymKeyCall_t encrypt;
    BYTE *iv;
    BYTE defaultIv[MAX_SYM_BLOCK_SIZE] = {0};
    // pAssert(dOut != NULL && key != NULL && dIn != NULL);
    if(dSize == 0)
        return TPM_RC_SUCCESS;
    TEST(algorithm);
    blockSize = CryptGetSymmetricBlockSize(algorithm, keySizeInBits);
    if(blockSize == 0)
        return TPM_RC_FAILURE;
    // If the iv is provided, then it is expected to be block sized. In some cases,
    // the caller is providing an array of 0's that is equal to [MAX_SYM_BLOCK_SIZE]
    // with no knowledge of the actual block size. This function will set it.
    if((ivInOut != NULL) && (mode != ALG_ECB_VALUE))
    {        ivInOut->t.size = blockSize;
        iv = ivInOut->t.buffer;
    }
    else        iv = defaultIv;
    pIv = iv;
    // Create encrypt key schedule and set the encryption function pointer.
    SELECT(ENCRYPT);
    switch(mode)
    {
    #if ALG_CTR
        case ALG_CTR_VALUE:
            for(; dSize > 0; dSize -= blockSize)
            {
                // Encrypt the current value of the IV(counter)
                ENCRYPT(&keySchedule, iv, tmp);
                //increment the counter (counter is big-endian so start at end)
```
for(i = blockSize - 1; i >= 0; i--)
  
  if((iv[i] += 1) != 0)
    
    break;

// XOR the encrypted counter value with input and put into output
pT = tmp;
for(i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
  
  *dOut++ = *dIn+ ^ *pT+;
}

break;
#endif

#if ALG_OFB
  
case ALG_OFB_VALUE:
    
    // This is written so that dIn and dOut may be the same
    for(; dSize > 0; dSize -= blockSize)
    
    {
      // Encrypt the current value of the "IV"
      ENCRYPT(&keySchedule, iv, iv);

      // XOR the encrypted IV into dIn to create the cipher text (dOut)
pIv = iv;
      for(i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
        
        *dOut++ = (*pIv++ ^ *dIn++);
    }

    break;
#endif

#if ALG_CBC
  
case ALG_CBC_VALUE:
    
    // For CBC the data size must be an even multiple of the cipher block size
    if((dSize % blockSize) != 0)
      return TPM_RC_SIZE;

    // XOR the data block into the IV, encrypt the IV into the IV
    // and then copy the IV to the output
    for(; dSize > 0; dSize -= blockSize)
    
    {
      pIv = iv;
      for(i = blockSize; i > 0; i--)
        
        *pIv++ ^= *dIn++;

      ENCRYPT(&keySchedule, iv, iv);

      pIv = iv;
      for(i = blockSize; i > 0; i--)
        
        *dOut++ = *pIv++;
    }

    break;
#endif

// CFB is not optional

#if ALG_CFB
  
case ALG_CFB_VALUE:
    
    // Encrypt the IV into the IV, XOR in the data, and copy to output
    for(; dSize > 0; dSize -= blockSize)
    
    {
      // Encrypt the current value of the IV
      ENCRYPT(&keySchedule, iv, iv);

      pIv = iv;
      for(i = (int)(dSize < blockSize) ? dSize : blockSize; i > 0; i--)
        
        // XOR the data into the IV to create the cipher text
        // and put into the output
        *dOut++ = *pIv++;

      // If the inner loop (i loop) was smaller than blockSize, then dSize
      // would have been smaller than blockSize and it is now negative. If
      // it is negative, then it indicates how many bytes are needed to pad
      // out the IV for the next round.
      for(; dSize < 0; dSize++)
        *pIv++ = 0;

    }

    break;
#if ALG_ECB
  case ALG_ECB_VALUE:
    // For ECB the data size must be an even multiple of the
    // cipher block size
    if((dSize % blockSize) != 0)
      return TPM_RC_SIZE;
    // Encrypt the input block to the output block
    for(; dSize > 0; dSize -= blockSize)
      {
        ENCRYPT(&keySchedule, dIn, dOut);
        dIn = &dIn[blockSize];
        dOut = &dOut[blockSize];
      }
    break;
  #endif
default:
    return TPM_RC_FAILURE;
  }
return TPM_RC_SUCCESS;
}

10.2.20.4.1 CryptSymmetricDecrypt()

This function performs symmetric decryption based on the mode.

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_FAILURE</td>
<td>A fatal error</td>
</tr>
<tr>
<td>TPM_RCS_SIZE</td>
<td>dSize is not a multiple of the block size for an algorithm that requires it</td>
</tr>
</tbody>
</table>

LIB_EXPORT TPM_RC
CryptSymmetricDecrypt(
BYTE *dOut,       // OUT: decrypted data
TPM_ALG_ID algorithm, // IN: the symmetric algorithm
UINT16 keySizeInBits, // IN: key size in bits
const BYTE *key,    // IN: key buffer. The size of this buffer
                     // in bytes is (keySizeInBits + 7) / 8
TPM2B_IV *ivInOut,  // IN/OUT: IV for decryption.
TPM_ALG_ID mode,    // IN: Mode to use
INT32 dSize,       // IN: data size (may need to be a
                     // multiple of the blockSize)
const BYTE *dIn     // IN: data buffer
                      )
{
BYTE *pIv;
int i;
BYTE tmp[MAX_SYM_BLOCK_SIZE];
BYTE *pt;
tpmCryptKeySchedule_t keySchedule;
INT16 blockSize;
BYTE *iv;
TpmCryptSetSymKeyCall_t encrypt;
TpmCryptSetSymKeyCall_t decrypt;
BYTE defaultIV[MAX_SYM_BLOCK_SIZE] = {0};

// These are used but the compiler can't tell because they are initialized
// in case statements and it can't tell if they are always initialized
// when needed, so... Comment these out if the compiler can tell or doesn't
// care that these are initialized before use.
encrypt = NULL;
decrypt = NULL;

pAssert(dOut != NULL && key != NULL && dIn != NULL);
if(dSize == 0)
    return TPM_RC_SUCCESS;

TEST(algorithm);
blockSize = CryptGetSymmetricBlockSize(algorithm, keySizeInBits);
if(blockSize == 0)
    return TPM_RC_FAILURE;

// If the iv is provided, then it is expected to be block sized. In some cases,
// the caller is providing an array of 0's that is equal to [MAX_SYM_BLOCK_SIZE]
// with no knowledge of the actual block size. This function will set it.
if((ivInOut != NULL) && (mode != ALG_ECB_VALUE))
{
    ivInOut->t.size = blockSize;
    iv = ivInOut->t.buffer;
}
else
    iv = defaultIv;

pIv = iv;

// Use the mode to select the key schedule to create. Encrypt always uses the
// encryption schedule. Depending on the mode, decryption might use either
// the decryption or encryption schedule.
switch(mode)
{
#if ALG_CBC || ALG_ECB
    case ALG_CBC_VALUE: // decrypt = decrypt
    case ALG_ECB_VALUE:
        // For ECB and CBC, the data size must be an even multiple of the
        // cipher block size
        if((dSize % blockSize) != 0)
            return TPM_RC_SIZE;
        SELECT(DECRYPT);
        break;
#endif
    default:
        // For the remaining stream ciphers, use encryption to decrypt
        SELECT(ENCRYPT);
        break;
#endif
    // Now do the mode-dependent decryption
    switch(mode)
    {
#if ALG_CBC
        case ALG_CBC_VALUE:
            // Copy the input data to a temp buffer, decrypt the buffer into the
            // output, XOR the IV, and copy the temp buffer to the IV and repeat.
            for(; dSize > 0; dSize -= blockSize)
            {
                pT = tmp;
                for(i = blockSize; i > 0; i--)
                    *pT++ = *dIn++;
                DECRYPT(&keySchedule, tmp, dOut);
                pIv = iv;
                pT = tmp;
                for(i = blockSize; i > 0; i--)
                {
                    *dOut++ ^= *pIv;
                    *pIv++ = *pT++;
                }
                break;
            }
#endif
            break;
#endif
        case ALG_CFB_VALUE:
            for(; dSize > 0; dSize -= blockSize)
            {
                pT = tmp;
                pIv = iv;
                pT = tmp;
                for(i = blockSize; i > 0; i--)
                {
                    *dOut++ ^= *pIv;
                    *pIv++ = *pT++;
                }
                break;
            }
}
323 // Encrypt the IV into the temp buffer
324 ENCRYPT(&keySchedule, iv, tmp);
325 pT = tmp;
326 pIV = iv;
327 for (i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
328 // Copy the current cipher text to IV, XOR
329 // with the temp buffer and put into the output
330 *dOut++ = *pT++ ^ (*pIV++ = *dIn++);
331 }
332 // If the inner loop (i loop) was smaller than blockSize, then dSize
333 // would have been smaller than blockSize and it is now negative
334 // If it is negative, then it indicates how may fill bytes
335 // are needed to pad out the IV for the next round.
336 for (; dSize < 0; dSize++)
337 *pIV++ = 0;
338 break;
339 #if ALG_CTR
340 case ALG_CTR_VALUE:
341 for (; dSize > 0; dSize -= blockSize)
342 // Encrypt the current value of the IV(counter)
343 ENCRYPT(&keySchedule, iv, tmp);
344 // Increment the counter (counter is big-endian so start at end)
345 for (i = blockSize - 1; i >= 0; i--)
346 if (iv[i] += 1) != 0
347 break;
348 // XOR the encrypted counter value with input and put into output
349 pT = tmp;
350 for (i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
351 *dOut++ = *dIn++ ^ *pT++;
352 }
353 break;
354 #endif
355 #if ALG_ECB
356 case ALG_ECB_VALUE:
357 for (; dSize > 0; dSize -= blockSize)
358 // Decrypt the current value of the IV
359 // Decrypt the current value of the IV
360 DECRIPT(&keySchedule, dIn, dOut);
361 dIn = &dIn[blockSize];
362 dOut = &dOut[blockSize];
363 }
364 break;
365 #endif
366 #if ALG_OFB
367 case ALG_OFB_VALUE:
368 // This is written so that dIn and dOut may be the same
369 for (; dSize > 0; dSize -= blockSize)
370 { // Encrypt the current value of the “IV”
371 ENCRYPT(&keySchedule, iv, iv);
372 // XOR the encrypted IV into dIn to create the cipher text (dOut)
373 pIV = iv;
374 for (i = (dSize < blockSize) ? dSize : blockSize; i > 0; i--)
375 *dOut++ = (*pIV++ ^ *dIn++);
376 }
377 break;
378 #endif
379 default:
380 return TPM_RC_FAILURE;
381 }
382 return TPM_RC_SUCCESS;
10.2.20.4.2 CryptSymKeyValidate()

Validate that a provided symmetric key meets the requirements of the TPM

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY_SIZE</td>
<td>Key size specifiers do not match</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>Key is not allowed</td>
</tr>
</tbody>
</table>

388 TPM_RC
389 CryptSymKeyValidate(
390     TPMT_SYM_DEF_OBJECT *symDef,
391     TPM2B_SYM_KEY       *key
392     )
393 {
394     if(key->t.size != BITS_TO_BYTES(symDef->keyBits.sym))
395         return TPM_RCS_KEY_SIZE;
396     #if ALG_TDES
397         if(symDef->algorithm == TPM_ALG_TDES && !CryptDesValidateKey(key))
398             return TPM_RCS_KEY;
399     #endif // ALG_TDES
400         return TPM_RC_SUCCESS;
401     }
10.2.21 PrimeData.c

```c
#include "Tpm.h"

This table is the product of all of the primes up to 1000. Checking to see if there is a GCD between a prime candidate and this number will eliminate many prime candidates from consideration before running Miller-Rabin on the result.

```c
const BN_STRUCT(43 * RADIX_BITS) s_CompositeOfSmallPrimes_ = {
    44, 44,
    { 0x2ED242696, 0x2BBFA177, 0x4820594F, 0xF73F4841,
      0x85FAC313A, 0x8C43E6B1, 0x6F262B68, 0x7FAB5061,
      0x59746F78, 0xF71377F6, 0x3B19B553, 0xBCDD0312,
      0x8B92E1FB, 0x3AC3152C, 0xE87C8273, 0xC0AE0E69,
      0x74A9E295, 0x448CCE86, 0x3C3A1907, 0x8A0BF944,
      0xF8C43B0E, 0xC26F0AF5, 0xC501C02F, 0x6579441A,
      0xD1099CDA, 0x6BC76A00, 0xC81A3228, 0xBB1AB25,
      0x70FA3841, 0x51BD07D6, 0xCC2359ED, 0xD9EE0769,
      0x75E47AFO, 0x045FF31E, 0x52CCE4F6, 0x04D8CB91,
      0x9665E8ED, 0x1753EFE5, 0x3AE4A5A6, 0x8FD4A97F,
      0x75E47AFO, 0x045FF31E, 0x52CCE4F6, 0x04D8CB91,
      0x9665E8ED, 0x1753EFE5, 0x3AE4A5A6, 0x8FD4A97F,
      0x0243C3E1, 0xE0F0C31D, 0x0000000B }
};

bigConst s_CompositeOfSmallPrimes = (const bigNum)&s_CompositeOfSmallPrimes_;
```

This table contains a bit for each of the odd values between 1 and $2^{16} + 1$. This table allows fast checking of the primes in that range. Don't change the size of this table unless you are prepared to do redo IsPrimeInt().
Trusted Platform Module Library

Part 4: Supporting Routines
#if RSA_KEY_SIEVE && SIMULATION && RSA_INSTRUMENT

UINT32 PrimeIndex = 0;
UINT32 failedAtIteration[10] = {0};
UINT32 PrimeCounts[3] = {0};
UINT32 MillerRabinTrials[3] = {0};
UINT32 totalFieldsSieved[3] = {0};
UINT32 bitsInFieldAfterSieve[3] = {0};
UINT32 emptyFieldsSieved[3] = {0};
UINT32 noPrimeFields[3] = {0};
UINT32 primesChecked[3] = {0};
UINT16 lastSievePrime = 0;
#endif
10.2.22 RsaKeyCache.c

10.2.22.1 Introduction

This file contains the functions to implement the RSA key cache that can be used to speed up simulation. Only one key is created for each supported key size and it is returned whenever a key of that size is requested.

If desired, the key cache can be populated from a file. This allows multiple TPM to run with the same RSA keys. Also, when doing simulation, the DRBG will use preset sequences so it is not too hard to repeat sequences for debug or profile or stress.

When the key cache is enabled, a call to CryptRsaGenerateKey() will call the GetCachedRsaKey(). If the cache is enabled and populated, then the cached key of the requested size is returned. If a key of the requested size is not available, the no key is loaded and the requested key will need to be generated. If the cache is not populated, the TPM will open a file that has the appropriate name for the type of keys required (CRT or no-CRT). If the file is the right size, it is used. If the file doesn't exist or the file does not have the correct size, the TMP will populate the cache with new keys of the required size and write the cache data to the file so that they will be available the next time.

Currently, if two simulations are being run with TPM's that have different RSA key sizes (e.g., one with 1024 and 2048 and another with 2048 and 3072, then the files will not match for the both of them and they will both try to overwrite the other's cache file. I may try to do something about this if necessary.

10.2.22.2 Includes, Types, Locals, and Defines

```c
#include "Tpm.h"
#include <stdio.h>
#include "Platform_fp.h"
#include "RsaKeyCache_fp.h"
#if CRT_FORMAT_RSA == YES
#define CACHE_FILE_NAME "RsaKeyCacheCrt.data"
#else
#define CACHE_FILE_NAME "RsaKeyCacheNoCrt.data"
#endif

typedef struct _RSA_KEY_CACHE_
{
    TPM2B_PUBLIC_KEY_RSA        publicModulus;
    TPM2B_PRIVATE_KEY_RSA       privatePrime;
    privateExponent_t           privateExponent;
} RSA_KEY_CACHE;
```

Determine the number of RSA key sizes for the cache

```c
#ifndef RSA_KEY_SIZE_BITS_1024
#define RSA_1024 YES
#else
#define RSA_1024 NO
#endif
#endif RSA_KEY_SIZE_BITS_2048
```
#ifdef RSA_KEY_SIZE_BITS_4096
#define RSA_4096 YES
#else
#define RSA_4096 NO
#endif

#define comma

TPMI_RSA_KEY_BITS SupportedRsaKeySizes[] = {
#if RSA_1024
1024
#else
#define comma ,
#endif
#if RSA_2048
comma 2048
#else
#define comma ,
#endif
#if RSA_3072
comma 3072
#else
#define comma ,
#endif
#if RSA_4096
comma 4096
#else
#endif
};
#define RSA_KEY_CACHE_ENTRIES (RSA_1024 + RSA_2048 + RSA_3072 + RSA_4096)

The key cache holds one entry for each of the supported key sizes.

RA_KEY_CACHE s_rsaKeyCache[RSA_KEY_CACHE_ENTRIES];
Indicates if the key cache is loaded. It can be loaded and enabled or disabled.

BOOL s_keyCacheLoaded = 0;
Indicates if the key cache is enabled.

int s_rsaKeyCacheEnabled = FALSE;

10.2.22.2.1 RsaKeyCacheControl()

Used to enable and disable the RSA key cache.

LIB_EXPORT void RsaKeyCacheControl(
    int state
)
{
    s_rsaKeyCacheEnabled = state;
}

10.2.22.2.2 InitializeKeyCache()

This will initialize the key cache and attempt to write it to a file for later use.
Return Value | Meaning
---|---
TRUE(1) | success
FALSE(0) | failure

```c
static BOOL
InitializeKeyCache(
    OBJECT *rsaKey,  // IN/OUT: The object structure in which
    RAND_STATE *rand  // IN: if not NULL, the deterministic
)
{
    int index;
    TPM_KEY_BITS keySave = rsaKey->publicArea.parameters.rsaDetail.keyBits;
    BOOL OK = TRUE;

    // s_rsaKeyCacheEnabled = FALSE;
    for(index = 0; OK && index < RSA_KEY_CACHE_ENTRIES; index++)
    {
        rsaKey->publicArea.parameters.rsaDetail.keyBits
            = SupportedRsaKeySizes[index];
        OK = (CryptRsaGenerateKey(rsaKey, rand) == TPM_RC_SUCCESS);
        if(OK)
        {
            s_rsaKeyCache[index].publicModulus = rsaKey->publicArea.unique.rsa;
            s_rsaKeyCache[index].privatePrime = rsaKey->sensitive.sensitive.rsa;
            s_rsaKeyCache[index].privateExponent = rsaKey->privateExponent;
        }
    }
    rsaKey->publicArea.parameters.rsaDetail.keyBits = keySave;
    s_keyCacheLoaded = OK;

    #if SIMULATION && USE_RSA_KEY_CACHE && USE_KEY_CACHE_FILE
    if(OK)
    {
        FILE *cacheFile;
        const char *fn = CACHE_FILE_NAME;

        #if defined _MSC_VER
        if(fopen_s(&cacheFile, fn, "w+b") != 0)
        #else
        cacheFile = fopen(fn, "w+b");
        if(NULL == cacheFile)
        #endif
        {
            printf("Can't open %s for write.\n", fn);
        }
        else
        {
            fseek(cacheFile, 0, SEEK_SET);
            if(fwrite(s_rsaKeyCache, 1, sizeof(s_rsaKeyCache), cacheFile)
                != sizeof(s_rsaKeyCache))
                printf("Error writing cache to %s.", fn);
        }
        if(cacheFile)
            fclose(cacheFile);
    }
    return s_keyCacheLoaded;
```
10.2.22.2.3 KeyCacheLoaded()

Checks that key cache is loaded.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>cache loaded</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>cache not loaded</td>
</tr>
</tbody>
</table>

```c
static BOOL KeyCacheLoaded(
    OBJECT *rsaKey,                      // IN/OUT: The object structure in which
    RAND_STATE *rand)                    // IN: if not NULL, the deterministic
{                                        // RNG state
    #if SIMULATION && USE_RSA_KEY_CACHE && USE_KEY_CACHE_FILE
        if(!s_keyCacheLoaded)
            {
                FILE *cacheFile;
                const char * fn = CACHE_FILE_NAME;
                #if defined _MSC_VER && 1
                    if(fopen_s(&cacheFile, fn, "r+b") == 0)
                #else
                    cacheFile = fopen(fn, "r+b");
                #endif
                if(NULL != cacheFile)
                    #endif
                    {
                        fseek(cacheFile, 0L, SEEK_END);
                        if(ftell(cacheFile) == sizeof(s_rsaKeyCache))
                            {
                                fseek(cacheFile, 0L, SEEK_SET);
                                s_keyCacheLoaded = {
                                    fread(&s_rsaKeyCache, 1, sizeof(s_rsaKeyCache), cacheFile)
                                    == sizeof(s_rsaKeyCache));
                            }
                        fclose(cacheFile);
                    }
        #endif

        if(!s_keyCacheLoaded)
            s_rsaKeyCacheEnabled = InitializeKeyCache(rsaKey, rand);
        return s_keyCacheLoaded;
    }
}
```

10.2.22.2.4 GetCachedRsaKey()

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>key loaded</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>key not loaded</td>
</tr>
</tbody>
</table>

```c
BOOL GetCachedRsaKey(
    OBJECT *key,                       // IN: if not NULL, the deterministic
    RAND_STATE *rand)                   // RNG state
{
    int keyBits = key->publicArea.parameters.rsaDetail.keyBits;
```
int index;

//
if(KeyCacheLoaded(key, rand))
{
    for(index = 0; index < RSA_KEY_CACHE_ENTRIES; index++)
    {
        if((s_rsaKeyCache[index].publicModulus.t.size * 8) == keyBits)
        {
            key->publicArea.unique.rsa = s_rsaKeyCache[index].publicModulus;
            key->sensitive.sensitive.rsa = s_rsaKeyCache[index].privatePrime;
            key->privateExponent = s_rsaKeyCache[index].privateExponent;
            key->attributes.privateExp = SET;
            return TRUE;
        }
    }
    return FALSE;
}

return s_keyCacheLoaded;

#endif // defined SIMULATION && defined USE_RSA_KEY_CACHE
10.2.23 Ticket.c

10.2.23.1 Introduction

This clause contains the functions used for ticket computations.

10.2.23.2 Includes

```c
#include "Tpm.h"
```

10.2.23.3 Functions

10.2.23.3.1 TicketIsSafe()

This function indicates if producing a ticket is safe. It checks if the leading bytes of an input buffer is TPM_GENERATED_VALUE or its substring of canonical form. If so, it is not safe to produce ticket for an input buffer claiming to be TPM generated buffer.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>safe to produce ticket</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>not safe to produce ticket</td>
</tr>
</tbody>
</table>

```c
BOOL TicketIsSafe(TPM2B *buffer)
{
    TPM_GENERATED valueToCompare = TPM_GENERATED_VALUE;
    BYTE bufferToCompare[sizeof(valueToCompare)];
    BYTE *marshalBuffer;

    // If the buffer size is less than the size of TPM_GENERATED_VALUE, assume
    // it is not safe to generate a ticket
    if(buffer->size < sizeof(valueToCompare))
        return FALSE;
    marshalBuffer = bufferToCompare;
    TPM_GENERATED_Marshal(valueToCompare, marshalBuffer, NULL);
    if(MemoryEqual(buffer->buffer, bufferToCompare, sizeof(valueToCompare)))
        return FALSE;
    else
        return TRUE;
}
```

10.2.23.3.2 TicketComputeVerified()

This function creates a TPMT_TK_VERIFIED ticket.

```c
void TicketComputeVerified(TPMI_RH_HIERARCHY hierarchy, // IN: hierarchy constant for ticket
                          TPM2B_DIGEST *digest, // IN: digest
                          TPM2B_NAME *keyName, // IN: name of key that signed the values
                          TPMT_TK_VERIFIED *ticket // OUT: verified ticket
                          )
{
    TPM2B_PROOF *proof;
    HMAC_STATE hmacState;
```
32 // Fill in ticket fields
33 ticket->tag = TPM_ST_VERIFIED;
34 ticket->hierarchy = hierarchy;
35 proof = HierarchyGetProof(hierarchy);
36
37 // Start HMAC using the proof value of the hierarchy as the HMAC key
38 ticket->digest.t.size = CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG,
39        &proof->b);
40 // TPM_ST_VERIFIED
41 CryptDigestUpdateInt(&hmacState, sizeof(TPM_ST), ticket->tag);
42 // digest
43 CryptDigestUpdate2B(&hmacState.hashState, &digest->b);
44 // key name
45 CryptDigestUpdate2B(&hmacState.hashState, &keyName->b);
46 // done
47 CryptHmacEnd2B(&hmacState, &ticket->digest.b);
48
49 return;
50 }

10.2.23.33 TicketComputeAuth()

This function creates a TPMT_TK_AUTH ticket.

52 void
53 TicketComputeAuth(
54      TPM_ST               type, // IN: the type of ticket.
55      TPMI_RH_HIERARCHY    hierarchy, // IN: hierarchy constant for ticket
56      UINT64               timeout, // IN: timeout
57      BOOL                expiresOnReset, // IN: flag to indicate if ticket expires on
58      //      TPM Reset
59      TPM2B_DIGEST        *cpHashA, // IN: input cpHashA
60      TPM2B_NONCE         *policyRef, // IN: input policyRef
61      TPM2B_NAME          *entityName, // IN: name of entity
62      TPMT_TK_AUTH        *ticket // OUT: Created ticket
63 )
64 {
65    TPM2B_PROOF         *proof;
66    HMAC_STATE           hmacState;
67    //
68    // Get proper proof
69    proof = HierarchyGetProof(hierarchy);
70    // Fill in ticket fields
71    ticket->tag = type;
72    ticket->hierarchy = hierarchy;
73
74    // Start HMAC with hierarchy proof as the HMAC key
75    ticket->digest.t.size = CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG,
76        &proof->b);
77    // TPM_ST_AUTH_SECRET or TPM_ST_AUTH_SIGNED,
78    CryptDigestUpdateInt(&hmacState, sizeof(UINT16), ticket->tag);
79    // cpHash
80    CryptDigestUpdate2B(&hmacState.hashState, &cpHashA->b);
81    // policyRef
82    CryptDigestUpdate2B(&hmacState.hashState, &policyRef->b);
83    // keyName
84    CryptDigestUpdate2B(&hmacState.hashState, &entityName->b);
85    // timeout
86    CryptDigestUpdateInt(&hmacState, sizeof(timeout), timeout);
87    if(timeout != 0)
88    {
89        // epoch
91  
92  CryptDigestUpdateInt(&hmacState.hashState, sizeof(CLOCK_NONCE),
93                     g_timeEpoch);
94  // reset count
95  if (expiresOnReset)
96    CryptDigestUpdateInt(&hmacState.hashState, sizeof(gp.totalResetCount),
97                        gp.totalResetCount);
98  }
99  // done
100  CryptHmacEnd2B(&hmacState, &ticket->digest.b);
101  return;
102 }

10.2.23.3.4 TicketComputeHashCheck()

This function creates a TPMT_TK_HASHCHECK ticket.

103 void TicketComputeHashCheck(
104     TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy constant for ticket
105     TPM_ALG_ID hashAlg,          // IN: the hash algorithm for 'digest'
106     TPM2B_DIGEST *digest,        // IN: input digest
107     TPMT_TK_HASHCHECK *ticket     // OUT: Created ticket
108 )
109 {
110   TPM2B_PROOF *proof;
111   HMAC_STATE hmacState;
112   // Get proper proof
113   proof = HierarchyGetProof(hierarchy);
114   // Fill in ticket fields
115   ticket->tag = TPM_ST_HASHCHECK;
116   ticket->hierarchy = hierarchy;
117   // Start HMAC using hierarchy proof as HMAC key
118   ticket->digest.t.size = CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG,
119                                             &proof->b);
120   // TPM_ST_HASHCHECK
121   CryptDigestUpdateInt(&hmacState, sizeof(TPM_ST), ticket->tag);
122   // hash algorithm
123   CryptDigestUpdateInt(&hmacState, sizeof(hashAlg), hashAlg);
124   // digest
125   CryptDigestUpdate2B(&hmacState.hashState, &digest->b);
126   // done
127   CryptHmacEnd2B(&hmacState, &ticket->digest.b);
128   return;
129 }

10.2.23.3.5 TicketComputeCreation()

This function creates a TPMT_TK_CREATION ticket.

135 void TicketComputeCreation(
136     TPMI_RH_HIERARCHY hierarchy,  // IN: hierarchy for ticket
137     TPM2B_NAME *name,             // IN: object name
138     TPM2B_DIGEST *creation,       // IN: creation hash
139     TPMT_TK_CREATION *ticket       // OUT: created ticket
140 )
141 {
142   TPM2B_PROOF *proof;
HMAC_STATE hmacState;

// Get proper proof
proof = HierarchyGetProof(hierarchy);

// Fill in ticket fields
ticket->tag = TPM_ST_CREATION;
ticket->hierarchy = hierarchy;

// Start HMAC using hierarchy proof as HMAC key
ticket->digest.t.size = CryptHmacStart2B(&hmacState, CONTEXT_INTEGRITY_HASH_ALG, &proof->b);

    // TPM_ST_CREATION
CryptDigestUpdateInt(&hmacState, sizeof(TPM_ST), ticket->tag);

    // name if provided
if (name != NULL)
    CryptDigestUpdate2B(&hmacState.hashState, &name->b);

    // creation hash
CryptDigestUpdate2B(&hmacState.hashState, &creation->b);

    // Done
CryptHmacEnd2B(&hmacState, &ticket->digest.b);

return;
Annex A
(informative)
Implementation Dependent

A.1 Introduction

This header file contains definitions that are used to define a TPM profile. Some of the values are derived from the values in the *TCG Algorithm Registry* [19] and others are simply chosen by the manufacturer. The values here are chosen to represent a full featured TPM so that all of the TPM’s capabilities can be simulated and tested. This file would change based on the implementation.

The file listed below was generated by an automated tool using three documents as inputs. They are:

a) The TCG_Algorthm Registry,

b) Part 2 of this specification, and

c) A purpose-built document that contains vendor-specific information in tables.

A.2 Implementation.h

```c
#ifndef _IMPLEMENTATION_H_
#define _IMPLEMENTATION_H_

#include "TpmBuildSwitches.h"
#include "BaseTypes.h"
#include "TPMB.h"

#undef TRUE
#undef FALSE
#undef MAX
#undef MIN

#define MAX(a, b) ((a) > (b) ? (a) : (b))

#define MIN(a, b) ((a) < (b) ? (a) : (b))

From TPM 2.0 Part 2: Table 4 - Defines for Logic Values

#define TRUE                1
#define FALSE               0
#define YES                 1
#define NO                  0
#define SET                 1
#define CLEAR               0

From Vendor-Specific: Table 1 - Defines for Processor Values

#define BIG_ENDIAN_TPM

#define BIG_ENDIAN_TPM

#define LITTLE_ENDIAN_TPM

#define MOST_SIGNIFICANT_BIT_0

#define MOST_SIGNIFICANT_BIT_0

#define LEAST_SIGNIFICANT_BIT_0

#define AUTO_ALIGN

#define AUTO_ALIGN

From Vendor-Specific: Table 3 - Defines for Key Size Constants

#define RSA_KEY_SIZES_BITS

#define RSA_KEY_SIZES_BITS
```

{1024,2048}
#define RSA_KEY_SIZE_BITS_1024          RSA_ALLOWED_KEY_SIZE_1024
#define RSA_KEY_SIZE_BITS_2048          RSA_ALLOWED_KEY_SIZE_2048
#define MAX_RSA_KEY_BITS                2048
#define MAX_RSA_KEY_BYTES               256
#define TDES_KEY_SIZES_BITS             {128,192}
#define TDES_KEY_SIZE_BITS_128          TDES_ALLOWED_KEY_SIZE_128
#define TDES_KEY_SIZE_BITS_192          TDES_ALLOWED_KEY_SIZE_192
#define MAX_TDES_KEY_BITS               192
#define MAX_TDES_KEY_BYTES              24
#define MAX_TDES_BLOCK_SIZE_BYTES       MAX(TDES_128_BLOCK_SIZE_BYTES, MAX(TDES_192_BLOCK_SIZE_BYTES, 0))
#define AES_KEY_SIZES_BITS              {128,256}
#define AES_KEY_SIZE_BITS_128           AES_ALLOWED_KEY_SIZE_128
#define AES_KEY_SIZE_BITS_256           AES_ALLOWED_KEY_SIZE_256
#define MAX_AES_KEY_BITS               256
#define MAX_AES_KEY_BYTES               32
#define MAX_AES_BLOCK_SIZE_BYTES       MAX(AES_128_BLOCK_SIZE_BYTES, MAX(AES_256_BLOCK_SIZE_BYTES, 0))
#define SM4_KEY_SIZES_BITS              {128}
#define SM4_KEY_SIZE_BITS_128           SM4_ALLOWED_KEY_SIZE_128
#define MAX_SM4_KEY_BITS               128
#define MAX_SM4_KEY_BYTES               16
#define MAX_SM4_BLOCK_SIZE_BYTES       MAX(SM4_128_BLOCK_SIZE_BYTES, 0)
#define CAMELLIA_KEY_SIZES_BITS         {128}
#define CAMELLIA_KEY_SIZE_BITS_128      CAMELLIA_ALLOWED_KEY_SIZE_128
#define MAX_CAMELLIA_KEY_BITS           128
#define MAX_CAMELLIA_KEY_BYTES          16
#define MAX_CAMELLIA_BLOCK_SIZE_BYTES   MAX(CAMELLIA_128_BLOCK_SIZE_BYTES, 0)
#define ECC_NIST_P192                   NO
#define ECC_NIST_P224                   NO
#define ECC_NIST_P256                   YES
#define ECC_NIST_P384                   YES
#define ECC_NIST_P521                   NO
#define ECC_BN_P256                     YES
#define ECC_BN_P638                     NO
#define ECC_SM2_P256                    NO
#define ECC_CURVES                      \  
{TPM_ECC_BN_P256, TPM_ECC_BN_P638, TPM_ECC_NIST_P192, \ 
  TPM_ECC_NIST_P224, TPM_ECC_NIST_P256, TPM_ECC_NIST_P384, \ 
  TPM_ECC_NIST_P521, TPM_ECC_SM2_P256}
#define ECC_CURVE_COUNT                 \  
{ECC_BN_P256 + ECC_BN_P638 + ECC_NIST_P192 + ECC_NIST_P224 + \ 
  ECC_NIST_P256 + ECC_NIST_P384 + ECC_NIST_P521 + ECC_SM2_P256}
#define MAX_ECC_KEY_BITS               \  
MAX(ECC_BN_P256 * 256, MAX(ECC_BN_P638 * 638, \ 
  MAX(ECC_NIST_P192 * 192, MAX(ECC_NIST_P224 * 224, \ 
  MAX(ECC_NIST_P256 * 256, MAX(ECC_NIST_P384 * 384, \ 
  MAX(ECC_NIST_P521 * 521, MAX(ECC_SM2_P256 * 256, \ 
  0))))))))
#define MAX_ECC_KEY_BYTES             BITS_TO_BYTES(MAX_ECC_KEY_BITS)
#define PLATFORM_FAMILY               TPM_SPEC_FAMILY
#define PLATFORM_LEVEL                TPM_SPEC_LEVEL
#define PLATFORM_VERSION              TPM_SPEC_VERSION
#define PLATFORM_YEAR                 TPM_SPEC_YEAR
#define PLATFORM_DAY_OF_YEAR          TPM_SPEC_DAY_OF_YEAR

From Vendor-Specific: Table 4 - Defines for Implemented Curves

From Vendor-Specific: Table 6 - Defines for PLATFORM Values

From Vendor-Specific: Table 7 - Defines for Implementation Values
#define FIELD_UPGRADEIMPLEMENTED NO
#define RADIX_BITS 32
#define HASH_ALIGNMENT 4
#define SYMMETRIC_ALIGNMENT 4
#define HASH_LIB OSSL
#define SYM_LIB OSSL
#define MATH_LIB OSSL
#define BSIZE UINT16
#define IMPLEMENTATION_PCR 24
#define PLATFORM_PCR 24
#define DRTM_PCR 17
#define HCRTM_PCR 0
#define NUM_LOCALITIES 5
#define MAX_HANDLE_NUM 3
#define MAX_ACTIVE_SESSIONS 64
#define CONTEXT_SLOT UINT16
#define CONTEXT_COUNTER UINT64
#define MAX_LOADED_SESSIONS 3
#define MAX_SESSION_NUM 3
#define MAX_LOADED_OBJECTS 3
#define MIN_EVICT_OBJECTS 2
#define NUM_POLICY_PCR_GROUP 1
#define NUM_AUTHVALUE_PCR_GROUP 1
#define MAX_CONTEXT_SIZE 2474
#define MAX_DIGEST_BUFFER 1024
#define MAX_NV_INDEX_SIZE 2048
#define MAX_NV_BUFFER_SIZE 1024
#define MAX_CAP_BUFFER 1024
#define NV_MEMORY_SIZE 16384
#define MIN_COUNTER_INDICES 8
#define MAX_ALG_LIST_SIZE 64
#define PRIMARY_SEED_SIZE 32
#define CONTEXT_ENCRYPT_ALGORITHM AES
#define NV_CLOCK_UPDATE_INTERVAL 12
#define MAX_COMMAND_SIZE 4096
#define MAX_RESPONSE_SIZE 4096
#define ORDERLY_BITS 8
#define MAX_SYM_DATA 128
#define MAX_RNG_ENTROPY_SIZE 64
#define RAM_INDEX_SPACE 512
#define RSA_DEFAULT_PUBLIC_EXPONENT 0x00010001
#define ENABLE_PCR_NO_INCREMENT YES
#define CRT_FORMAT_RSA YES
#define VENDOR_COMMAND_COUNT 0
#define MAX_VENDOR_BUFFER_SIZE 1024
#define TPM_MAX_DERIVATION_BITS 8192

From Vendor-Specific: Table 2 - Defines for Implemented Algorithms

#define ALG_AES ALG_YES
#define ALG_CAMELLIA ALG_NO /* Not specified by vendor */
#define ALG_CBC ALG_YES
#define ALG_CFB ALG_YES
#define ALG_CMAC ALG_YES
#define ALG_CTR ALG_YES
#define ALG_ECB ALG_YES
#define ALG_ECC ALG_YES
#define ALG_EDDA (ALG_YES && ALG_ECC)
#define ALG_ECDH (ALG_YES && ALG_ECC)
#define ALG_ECDSA (ALG_YES && ALG_ECC)
#define ALG_ECMQV (ALG_NO && ALG_ECC)
#define ALG_EC Schnorr (ALG_YES && ALG_ECC)
#define ALG_HMAC ALG_YES
149 #define ALG_KDF1_SP800_108 ALG_YES
150 #define ALG_KDF1_SP800_56A (ALG_YES && ALG_ECC)
151 #define ALG_KDF2 ALG_NO
152 #define ALG_KEYEDHASH ALG_YES
153 #define ALG_MGF1 ALG_YES
154 #define ALG_OAEP (ALG_YES && ALG_RSA)
155 #define ALG_OFB ALG_YES
156 #define ALG_RSA ALG_YES
157 #define ALG_RSAE (ALG_YES && ALG_RSA)
158 #define ALG_RSAPE (ALG_YES && ALG_RSA)
159 #define ALG_RSASSA (ALG_YES && ALG_RSA)
160 #define ALG_SHA ALG_NO /* Not specified by vendor */
161 #define ALG_SHA1 ALG_YES
162 #define ALG_SHA256 ALG_YES
163 #define ALG_SHA384 ALG_YES
164 #define ALG_SHA512 ALG_NO
165 #define ALG_SM2 (ALG_NO && ALG_ECC)
166 #define ALG_SM3_256 ALG_NO
167 #define ALG_SM4 ALG_NO
168 #define ALG_SYMCIPHER ALG_YES
169 #define ALG_XOR ALG_YES
170 #define ALG_HEADER TABLE - Definition of TPM_ALG_ID Constants

171 typedef UINT16 TPM_ALG_ID;
172 #define ALG_ERROR_VALUE 0x0000
173 #define TPM_ALG_ERROR (TPM_ALG_ID)(ALG_ERROR_VALUE)
174 #define ALG_RSA_VALUE 0x0003
175 #if ALG_RSA
176 #define TPM_ALG_RSA (TPM_ALG_ID)(ALG_RSA_VALUE)
177 #endif
178 #define ALG_TDES VALUE 0x0004
179 #if ALG_TDES
180 #define TPM_ALG_TDES (TPM_ALG_ID)(ALG_TDES_VALUE)
181 #endif
182 #define ALG_SHA VALUE 0x0005
183 #if ALG_SHA
184 #define TPM_ALG_SHA (TPM_ALG_ID)(ALG_SHA_VALUE)
185 #endif
186 #define ALG_SHA1 VALUE 0x000A
187 #if ALG_SHA1
188 #define TPM_ALG_SHA1 (TPM_ALG_ID)(ALG_SHA1_VALUE)
189 #endif
190 #define ALG_HMAC VALUE 0x0006
191 #if ALG_HMAC
192 #define TPM_ALG_HMAC (TPM_ALG_ID)(ALG_HMAC_VALUE)
193 #endif
194 #define ALG_AES VALUE 0x0007
195 #if ALG_AES
196 #define TPM_ALG_AES (TPM_ALG_ID)(ALG_AES_VALUE)
197 #endif
198 #define TPM_ALG_MGF1 VALUE 0x0008
199 #if ALG_MGF1
200 #define TPM_ALG_MGF1 (TPM_ALG_ID)(ALG_MGF1_VALUE)
201 #endif
202 #define TPM_ALG_KEYEDHASH VALUE 0x0009
203 #if ALG_KEYEDHASH
204 #define TPM_ALG_KEYEDHASH (TPM_ALG_ID)(ALG_KEYEDHASH_VALUE)
205 #endif
206 #define TPM_ALG_XOR VALUE 0x000A
207 #if ALG_XOR
208 #define TPM_ALG_XOR (TPM_ALG_ID)(ALG_XOR_VALUE)
209 #endif
210 #define TPM_ALG_SHA256 VALUE 0x000B
#if ALG_SHA256
#define TPM_ALG_SHA256 (TPM_ALG_ID)(ALG_SHA256_VALUE)
#endif // ALG_SHA256

#define ALG_SHA384_VALUE            0x000C
#if ALG_SHA384
#define TPM_ALG_SHA384 (TPM_ALG_ID)(ALG_SHA384_VALUE)
#endif // ALG_SHA384

#define ALG_SHA512_VALUE            0x000D
#if ALG_SHA512
#define TPM_ALG_SHA512 (TPM_ALG_ID)(ALG_SHA512_VALUE)
#endif // ALG_SHA512

#define ALG_NULL_VALUE              0x0010
#define TPM_ALG_NULL (TPM_ALG_ID)(ALG_NULL_VALUE)

#define ALG_SM3_256_VALUE           0x0012
#if ALG_SM3_256
#define TPM_ALG_SM3_256 (TPM_ALG_ID)(ALG_SM3_256_VALUE)
#endif // ALG_SM3_256

#define ALG_SM4_VALUE               0x0013
#if ALG_SM4
#define TPM_ALG_SM4 (TPM_ALG_ID)(ALG_SM4_VALUE)
#endif // ALG_SM4

#define ALG_RSASSA_VALUE            0x0014
#if ALG_RSASSA
#define TPM_ALG_RSASSA (TPM_ALG_ID)(ALG_RSASSA_VALUE)
#endif // ALG_RSASSA

#define ALG_RSAES_VALUE             0x0015
#if ALG_RSAES
#define TPM_ALG_RSAES (TPM_ALG_ID)(ALG_RSAES_VALUE)
#endif // ALG_RSAES

#define ALG_RSAPSS_VALUE            0x0016
#if ALG_RSAPSS
#define TPM_ALG_RSAPSS (TPM_ALG_ID)(ALG_RSAPSS_VALUE)
#endif // ALG_RSAPSS

#define ALG_OAEPP_VALUE              0x0017
#if ALG_OAEPP
#define TPM_ALG_OAEPP (TPM_ALG_ID)(ALG_OAEPP_VALUE)
#endif // ALG_OAEPP

#define ALG_ECDSA_VALUE             0x0018
#if ALG_ECDSA
#define TPM_ALG_ECDSA (TPM_ALG_ID)(ALG_ECDSA_VALUE)
#endif // ALG_ECDSA

#define ALG_ECDH_VALUE              0x0019
#if ALG_ECDH
#define TPM_ALG_ECDH (TPM_ALG_ID)(ALG_ECDH_VALUE)
#endif // ALG_ECDH

#define ALG_ECDSAADM_VALUE           0x001A
#if ALG_ECDSAADM
#define TPM_ALG_ECDSAADM (TPM_ALG_ID)(ALG_ECDSAADM_VALUE)
#endif // ALG_ECDSAADM

#define ALG_SM2_VALUE               0x001B
#if ALG_SM2
#define TPM_ALG_SM2 (TPM_ALG_ID)(ALG_SM2_VALUE)
#endif // ALG_SM2

#define ALG_ECSCNORR_VALUE           0x001C
#if ALG_ECSCNORR
#define TPM_ALG_ECSCNORR (TPM_ALG_ID)(ALG_ECSCNORR_VALUE)
#endif // ALG_ECSCNORR

#define ALG_ECMQV_VALUE              0x001D
#if ALG_ECMQV
#define TPM_ALG_ECMQV (TPM_ALG_ID)(ALG_ECMQV_VALUE)
#endif // ALG_ECMQV

#define ALG_KDF1_SP800_56A_VALUE     0x0020
#if ALG_KDF1_SP800_56A
#define TPM_ALG_KDF1_SP800_56A (TPM_ALG_ID)(ALG_KDF1_SP800_56A_VALUE)
#endif // ALG_KDF1_SP800_56A

#define ALG_KDF2_VALUE              0x0021
#if ALG_KDF2
#define TPM_ALG_KDF2 (TPM_ALG_ID)(ALG_KDF2_VALUE)
#endif // ALG_KDF2

#if ALG_KDF1_SP800_108
#define TPM_ALG_KDF1_SP800_108 (TPM_ALG_ID)(ALG_KDF1_SP800_108_VALUE)
#endif // ALG_KDF1_SP800_108

#if ALG_ECC
#define TPM_ALG_ECC (TPM_ALG_ID)(ALG_ECC_VALUE)
#endif // ALG_ECC

#if ALG_SYMCIPHER
#define TPM_ALG_SYMCIPHER (TPM_ALG_ID)(ALG_SYMCIPHER_VALUE)
#endif // ALG_SYMCIPHER

#if ALG_CAMELLIA
#define TPM_ALG_CAMELLIA (TPM_ALG_ID)(ALG_CAMELLIA_VALUE)
#endif // ALG_CAMELLIA

#if ALG_CMAC
#define TPM_ALG_CMAC (TPM_ALG_ID)(ALG_CMAC_VALUE)
#endif // ALG_CMAC

#if ALG_CTR
#define TPM_ALG_CTR (TPM_ALG_ID)(ALG_CTR_VALUE)
#endif // ALG_CTR

#if ALG_OFB
#define TPM_ALG_OFB (TPM_ALG_ID)(ALG_OFB_VALUE)
#endif // ALG_OFB

#if ALG_CBC
#define TPM_ALG_CBC (TPM_ALG_ID)(ALG_CBC_VALUE)
#endif // ALG_CBC

#if ALG_CFB
#define TPM_ALG_CFB (TPM_ALG_ID)(ALG_CFB_VALUE)
#endif // ALG_CFB

#if ALG_ECB
#define TPM_ALG_ECB (TPM_ALG_ID)(ALG_ECB_VALUE)
#endif // ALG_ECB

Values derived from Table 1:2

#define ALG_FIRST_VALUE 0x0001
#define TPM_ALG_FIRST (TPM_ALG_ID)(ALG_FIRST_VALUE)
#define ALG_LAST_VALUE 0x0044
#define TPM_ALG_LAST (TPM_ALG_ID)(ALG_LAST_VALUE)

From TCG Algorithm Registry: Table 3 - Definition of TPM_ECC_CURVE Constants

typedef UINT16 TPM_ECC_CURVE;
#define TPM_ECC_NONE (TPM_ECC_CURVE)(0x0000)
#define TPM_ECC_NIST_P192 (TPM_ECC_CURVE)(0x0001)
#define TPM_ECC_NIST_P224 (TPM_ECC_CURVE)(0x0002)
#define TPM_ECC_NIST_P256 (TPM_ECC_CURVE)(0x0003)
#define TPM_ECC_NIST_P384 (TPM_ECC_CURVE)(0x0004)
#define TPM_ECC_NIST_P521 (TPM_ECC_CURVE)(0x0005)
#define TPM_ECC_BN_P256 (TPM_ECC_CURVE)(0x0010)
#define TPM_ECC_BN_P638 (TPM_ECC_CURVE)(0x0011)
#define TPM_ECC_SM2_P256 (TPM_ECC_CURVE)(0x0020)
Part 4: Supporting Routines

From TCG Algorithm Registry: Table 12 - Defines for SHA1 Hash Values

334  #define SHA1_DIGEST_SIZE    20
335  #define SHA1_BLOCK_SIZE     64
336  #define SHA1_DER_SIZE       15
337  #define SHA1_DER
338     0x30, 0x21, 0x30, 0x09, 0x06, 0x05, 0x2B, 0x0E,
339           0x03, 0x02, 0x1A, 0x05, 0x00, 0x04, 0x14

From TCG Algorithm Registry: Table 13 - Defines for SHA256 Hash Values

340  #define SHA256_DIGEST_SIZE  32
341  #define SHA256_BLOCK_SIZE   64
342  #define SHA256_DER_SIZE     19
343  #define SHA256_DER
344     0x30, 0x31, 0x30, 0x0D, 0x06, 0x09, 0x60, 0x86,
345           0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x01, 0x05,
346             0x00, 0x04, 0x20

From TCG Algorithm Registry: Table 14 - Defines for SHA384 Hash Values

347  #define SHA384_DIGEST_SIZE  48
348  #define SHA384_BLOCK_SIZE   128
349  #define SHA384_DER_SIZE     19
350  #define SHA384_DER
351     0x30, 0x41, 0x30, 0x0D, 0x06, 0x09, 0x60, 0x86,
352           0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x02, 0x05,
353             0x00, 0x04, 0x30

From TCG Algorithm Registry: Table 15 - Defines for SHA512 Hash Values

354  #define SHA512_DIGEST_SIZE  64
355  #define SHA512_BLOCK_SIZE   128
356  #define SHA512_DER_SIZE     19
357  #define SHA512_DER
358     0x30, 0x51, 0x30, 0x0D, 0x06, 0x09, 0x60, 0x86,
359           0x48, 0x01, 0x65, 0x03, 0x04, 0x02, 0x03, 0x05,
360             0x00, 0x04, 0x40

From TCG Algorithm Registry: Table 16 - Defines for SM3_256 Hash Values

361  #define SM3_256_DIGEST_SIZE     32
362  #define SM3_256_BLOCK_SIZE      64
363  #define SM3_256_DER_SIZE        18
364  #define SM3_256_DER
365     0x30, 0x30, 0x30, 0x0C, 0x06, 0x08, 0x2A, 0x81,
366           0x1C, 0x81, 0x45, 0x01, 0x83, 0x11, 0x05, 0x00,
367             0x04, 0x20

From TCG Algorithm Registry: Table 17 - Defines for AES Symmetric Cipher Algorithm Constants

368  #define AES_ALLOWED_KEY_SIZE_128    YES
369  #define AES_ALLOWED_KEY_SIZE_192    YES
370  #define AES_ALLOWED_KEY_SIZE_256    YES
371  #define AES_128_BLOCK_SIZE_BYTES    16
372  #define AES_192_BLOCK_SIZE_BYTES    16
373  #define AES_256_BLOCK_SIZE_BYTES    16

From TCG Algorithm Registry: Table 18 - Defines for SM4 Symmetric Cipher Algorithm Constants

374  #define SM4_ALLOWED_KEY_SIZE_128    YES
375  #define SM4_ALLOWED_KEY_SIZE_128    YES
From TCG Algorithm Registry: Table 19 - Defines for CAMELLIA Symmetric Cipher Algorithm Constants

376  #define CAMELLIA_ALLOWED_KEY_SIZE_128 YES
377  #define CAMELLIA_ALLOWED_KEY_SIZE_192 YES
378  #define CAMELLIA_ALLOWED_KEY_SIZE_256 YES
379  #define CAMELLIA_128_BLOCK_SIZE_BYTES 16
380  #define CAMELLIA_192_BLOCK_SIZE_BYTES 16
381  #define CAMELLIA_256_BLOCK_SIZE_BYTES 16

From TCG Algorithm Registry: Table 17 - Defines for TDES Symmetric Cipher Algorithm Constants

382  #define TDES_ALLOWED_KEY_SIZE_128 YES
383  #define TDES_ALLOWED_KEY_SIZE_192 YES
384  #define TDES_128_BLOCK_SIZE_BYTES 8
385  #define TDES_192_BLOCK_SIZE_BYTES 8

From Vendor-Specific: Table 5 - Defines for Implemented Commands

386  #define CC_AC_GetCapability       CC_YES
387  #define CC_AC_Send                CC_YES
388  #define CC_ActivateCredential    CC_YES
389  #define CC_Certify               CC_YES
390  #define CC_CertifyCreation       CC_YES
391  #define CC_ChangeEPS             CC_YES
392  #define CC_ChangePPS             CC_YES
393  #define CC_Clear                 CC_YES
394  #define CC_ClearControl          CC_YES
395  #define CC_ClockRateAdjust       CC_YES
396  #define CC_ClockSet               CC_YES
397  #define CC_Commit                 (CC_YES && ALG_ECC)
398  #define CC_ContextLoad            CC_YES
399  #define CC_ContextSave            CC_YES
400  #define CC_Create                CC_YES
401  #define CC_CreateLoaded           CC_YES
402  #define CC_CreatePrimary          CC_YES
403  #define CC_DictionaryAttackLockReset CC_YES
404  #define CC_DictionaryAttackParameters CC_YES
405  #define CC_Duplicate              CC_YES
406  #define CC_ECC_Parameters         (CC_YES && ALG_ECC)
407  #define CC_ECDH_KeyGen            (CC_YES && ALG_ECC)
408  #define CC_ECDH_ZGen               (CC_YES && ALG_ECC)
409  #define CC_EC_Ephemeral           (CC_YES && ALG_ECC)
410  #define CC_EncryptDecrypt         CC_YES
411  #define CC_EncryptDecrypt2       CC_YES
412  #define CC_EventSequenceComplete CC_YES
413  #define CC_EvictControl           CC_YES
414  #define CC_FieldUpgradeData      CC_NO
415  #define CC_FieldUpgradeStart     CC_NO
416  #define CC_FirmwareRead           CC_NO
417  #define CC_FlushContext           CC_YES
418  #define CC_GetCapability          CC_YES
419  #define CC_GetCommandAuditDigest CC_YES
420  #define CC_GetRandom              CC_YES
421  #define CC_GetSessionAuditDigest CC_YES
422  #define CC_GetTestResult          CC_YES
423  #define CC_HMAC                   (CC_YES && !ALG_CMAC)
424  #define CC_HMAC_Start              (CC_YES && !ALG_CMAC)
425  #define CC_HMAC_Start              (CC_YES && !ALG_CMAC)
426  #define CC_Hash                   CC_YES
427  #define CC_HashSequenceStart      CC_YES
428  #define CC_HierarchyChangeAuth    CC_YES
429  #define CC_HierarchyControl       CC_YES
430  #define CC_Import                 CC_YES
431  #define CC_IncrementalSelfTest    CC_YES
#define CC_Load CC_YES
#define CC_LoadExternal CC_YES
#define CC_MAC (CC_YES && ALG_CMAC)
#define CC_MAC_Start (CC_YES && ALG_CMAC)
#define CC_MakeCredential CC_YES
#define CC_NV_Certify CC_YES
#define CC_NV_ChangeAuth CC_YES
#define CC_NV_DefineSpace CC_YES
#define CC_NV_Extend CC_YES
#define CC_NV_GlobalWriteLock CC_YES
#define CC_NV_Increment CC_YES
#define CC_NV_Read CC_YES
#define CC_NV_ReadLock CC_YES
#define CC_NV_ReadPublic CC_YES
#define CC_NV_SetBits CC_YES
#define CC_NV_UndefineSpace CC_YES
#define CC_NV_UndefineSpaceSpecial CC_YES
#define CC_NV_Write CC_YES
#define CC_NV_WriteLock CC_YES
#define CC_ObjectChangeAuth CC_YES
#define CC_PCR_Allocate CC_YES
#define CC_PCR_Event CC_YES
#define CC_PCR_Extend CC_YES
#define CC_PCR_Read CC_YES
#define CC_PCR_Reset CC_YES
#define CC_PCR_SetAuthPolicy CC_YES
#define CC_PCR_SetAuthValue CC_YES
#define CC_PP_Commands CC_YES
#define CC_PolicyAuthValue CC_YES
#define CC_PolicyAuthorize CC_YES
#define CC_PolicyAuthorizeNV CC_YES
#define CC_PolicyCommandCode CC_YES
#define CC_PolicyCounterTimer CC_YES
#define CC_PolicyCpHash CC_YES
#define CC_PolicyDuplicationSelect CC_YES
#define CC_PolicyGetDigest CC_YES
#define CC_PolicyLocality CC_YES
#define CC_PolicyNV CC_YES
#define CC_PolicyNameHash CC_YES
#define CC_PolicyNVWritten CC_YES
#define CC_PolicyOR CC_YES
#define CC_PolicyPCR CC_YES
#define CC_PolicyPassword CC_YES
#define CC_PolicyPhysicalPresence CC_YES
#define CC_PolicyRestart CC_YES
#define CC_PolicySecret CC_YES
#define CC_PolicySigned CC_YES
#define CC_PolicyTemplate CC_YES
#define CC_PolicyTicket CC_YES
#define CC_Policy_AC_SendSelect CC_YES
#define CC_Qoute CC_YES
#define CC_RSA_Decrypt (CC_YES && ALG_RSA)
#define CC_RSA_Encrypt (CC_YES && ALG_RSA)
#define CC_ReadClock CC_YES
#define CC_ReadPublic CC_YES
#define CC_Rewrap CC_YES
#define CC_SelfTest CC_YES
#define CC_SequenceComplete CC_YES
#define CC_SequenceUpdate CC_YES
#define CC_SetAlgorithmSet CC_YES
#define CC_SetCommandCodeAuditStatus CC_YES
#define CC_SetPrimaryPolicy CC_YES
#define CC_Shutdown CC_YES
#define CC_Sign CC_YES
#define CC_StartAuthSession CC_YES
#define CC_Startup CC_YES
#define CC_StirRandom                      CC_YES
#define CC_TestParms                        CC_YES
#define CC_Unseal                           CC_YES
#define CC_Vendor_TCG_Test                  CC_YES
#define CC_VerifySignature                  CC_YES
#define CC_ZGen_2Phase                      (CC_YES && ALG_ECC)

From TPM 2.0 Part 2: Table 12 - Definition of TPM_CC Constants

typedef UINT32                              TPM_CC;
#if CC_NV_UndefineSpaceSpecial
#define TPM_CC_NV_UndefineSpaceSpecial      (TPM_CC)(0x0000011F)
#endif
#if CC_EvictControl
#define TPM_CC_EvictControl                 (TPM_CC)(0x00000120)
#endif
#if CC_HierarchyControl
#define TPM_CC_HierarchyControl             (TPM_CC)(0x00000121)
#endif
#if CC_NV_UndefineSpace
#define TPM_CC_NV_UndefineSpace             (TPM_CC)(0x00000122)
#endif
#if CC_ChangeEPS
#define TPM_CC_ChangeEPS                    (TPM_CC)(0x00000124)
#endif
#if CC_ChangePPS
#define TPM_CC_ChangePPS                    (TPM_CC)(0x00000125)
#endif
#if CC_Clear
#define TPM_CC_Clear                        (TPM_CC)(0x00000126)
#endif
#if CC_ClearControl
#define TPM_CC_ClearControl                 (TPM_CC)(0x00000127)
#endif
#if CC_ClockSet
#define TPM_CC_ClockSet
#endif
#if CC_HierarchyChangeAuth
#define TPM_CC_HierarchyChangeAuth          (TPM_CC)(0x00000129)
#endif
#if CC_NV_DefineSpace
#define TPM_CC_NV_DefineSpace               (TPM_CC)(0x0000012A)
#endif
#if CC_PCR_Allocate
#define TPM_CC_PCR_Allocate                 (TPM_CC)(0x0000012B)
#endif
#if CC_PCR_SetAuthPolicy
#define TPM_CC_PCR_SetAuthPolicy            (TPM_CC)(0x0000012C)
#endif
#if CC_PP_Commands
#define TPM_CC_PP_Commands                  (TPM_CC)(0x0000012D)
#endif
#if CC_SetPrimaryPolicy
#define TPM_CC_SetPrimaryPolicy             (TPM_CC)(0x0000012E)
#endif
#if CC_FieldUpgradeStart
#define TPM_CC_FieldUpgradeStart            (TPM_CC)(0x0000012F)
#endif
#if CC_ClockRateAdjust
#define TPM_CC_ClockRateAdjust              (TPM_CC)(0x00000130)
#endif
#if CC_CreatePrimary
#define TPM_CC_CreatePrimary                (TPM_CC)(0x00000131)
#endif
#if CC_NV_GlobalWriteLock
#define TPM_CC_NV_GlobalWriteLock        (TPM_CC)(0x00000132)
#endif
#if
#define TPM_CC_NV_GlobalWriteLock        (TPM_CC)(0x00000132)
#endif
#endif
#define TPM_CC_NV_Increment             (TPM_CC)(0x00000134)
#endif
#define TPM_CC_NV_SetBits               (TPM_CC)(0x00000135)
#endif
#define TPM_CC_NV_Extend                (TPM_CC)(0x00000136)
#endif
#define TPM_CC_NV_WriteLock             (TPM_CC)(0x00000138)
#endif
#define TPM_CC_DictionaryAttackLockReset (TPM_CC)(0x00000139)
#endif
#define TPM_CC_DictionaryAttackParameters (TPM_CC)(0x0000013A)
#endif
#define TPM_CC_NV_ChangeAuth            (TPM_CC)(0x0000013B)
#endif
#define TPM_CC_PCR_Event                (TPM_CC)(0x0000013C)
#endif
#define TPM_CC_PCR_Reset                (TPM_CC)(0x0000013D)
#endif
#define TPM_CC_SequenceComplete         (TPM_CC)(0x0000013E)
#endif
#define TPM_CC_SetAlgorithmSet          (TPM_CC)(0x0000013F)
#endif
#define TPM_CC_SetCommandCodeAuditStatus (TPM_CC)(0x00000140)
#endif
#define TPM_CC_FieldUpgradeData          (TPM_CC)(0x00000141)
#endif
#define TPM_CC_IncrementalSelfTest       (TPM_CC)(0x00000142)
#endif
#define TPM_CC_SelfTest                  (TPM_CC)(0x00000143)
#endif
#define TPM_CC_Startup                   (TPM_CC)(0x00000144)
#endif
#define TPM_CC_Shutdown                  (TPM_CC)(0x00000145)
#endif
#define TPM_CC_StirRandom                (TPM_CC)(0x00000146)
#endif
#define TPM_CC_ActivateCredential        (TPM_CC)(0x00000147)
#endif
#define TPM_CC_Certify
#define TPM_CC_Certify                      (TPM_CC)(0x00000148)
#endif
#if CC_PolicyNV
#define TPM_CC_PolicyNV                     (TPM_CC)(0x00000149)
#endif
#if CC_CertifyCreation
#define TPM_CC_CertifyCreation              (TPM_CC)(0x0000014A)
#endif
#if CC_Duplicate
#define TPM_CC_Duplicate                    (TPM_CC)(0x0000014B)
#endif
#if CC_GetTime
#define TPM_CC_GetTime                      (TPM_CC)(0x0000014C)
#endif
#if CC_GetSessionAuditDigest
#define TPM_CC_GetSessionAuditDigest        (TPM_CC)(0x0000014D)
#endif
#if CC_NV_Read
#define TPM_CC_NV_Read                      (TPM_CC)(0x0000014E)
#endif
#if CC_NV_ReadLock
#define TPM_CC_NV_ReadLock                  (TPM_CC)(0x0000014F)
#endif
#if CC_ObjectChangeAuth
#define TPM_CC_ObjectChangeAuth             (TPM_CC)(0x00000150)
#endif
#if CC_PolicySecret
#define TPM_CC_PolicySecret                 (TPM_CC)(0x00000151)
#endif
#if CC_Rewrap
#define TPM_CC_Rewrap
#endif
#if CC_Create
#define TPM_CC_Create                       (TPM_CC)(0x00000153)
#endif
#if CC_ECDH_ZGen
#define TPM_CC_ECDH_ZGen                    (TPM_CC)(0x00000154)
#endif
#if CC_HMAC
#define TPM_CC_HMAC                         (TPM_CC)(0x00000155)
#endif
#if CC_MAC
#define TPM_CC_MAC                          (TPM_CC)(0x00000155)
#endif
#if CC_Import
#define TPM_CC_Import                       (TPM_CC)(0x00000156)
#endif
#if CC_Load
#define TPM_CC_Load                         (TPM_CC)(0x00000157)
#endif
#if CC_Quote
#define TPM_CC_Quote                        (TPM_CC)(0x00000158)
#endif
#if CC_RSA_Decrypt
#define TPM_CC_RSA_Decrypt                  (TPM_CC)(0x00000159)
#endif
#if CC_HMAC_Start
#define TPM_CC_HMAC_Start                   (TPM_CC)(0x0000015B)
#endif
#if CC_MAC_Start
#define TPM_CC_MAC_Start                    (TPM_CC)(0x0000015B)
#endif
#if CC_SequenceUpdate
#define TPM_CC_SequenceUpdate               (TPM_CC)(0x0000015C)
#endif
#if CC_Sign
#define TPM_CC_SIGN                         (TPM_CC)(0x0000015D)
#endif
#if CC_Unseal
#define TPM_CC_Unseal (TPM_CC)(0x0000015E)
#endif
#if CC_PolicySigned
#define TPM_CC_PolicySigned (TPM_CC)(0x00000160)
#endif
#if CC_ContextLoad
#define TPM_CC_ContextLoad (TPM_CC)(0x00000161)
#endif
#if CC_ContextSave
#define TPM_CC_ContextSave (TPM_CC)(0x00000162)
#endif
#if CC_ECDH.KeyGen
#define TPM_CC_ECDH_KeyGen (TPM_CC)(0x00000163)
#endif
#if CC_EncryptDecrypt
#define TPM_CC_EncryptDecrypt (TPM_CC)(0x00000164)
#endif
#if CC_FlushContext
#define TPM_CC_FlushContext (TPM_CC)(0x00000165)
#endif
#if CC_LoadExternal
#define TPM_CC_LoadExternal (TPM_CC)(0x00000167)
#endif
#if CC_MakeCredential
#define TPM_CC_MakeCredential (TPM_CC)(0x00000168)
#endif
#if CC_NV_ReadPublic
#define TPM_CC_NV_ReadPublic (TPM_CC)(0x00000169)
#endif
#if CC_PolicyAuthorize
#define TPM_CC_PolicyAuthorize (TPM_CC)(0x0000016A)
#endif
#if CC_PolicyAuthValue
#define TPM_CC_PolicyAuthValue (TPM_CC)(0x0000016B)
#endif
#if CC_PolicyCommandCode
#define TPM_CC_PolicyCommandCode (TPM_CC)(0x0000016C)
#endif
#if CC_PolicyCounterTimer
#define TPM_CC_PolicyCounterTimer (TPM_CC)(0x0000016D)
#endif
#if CC_PolicyCpHash
#define TPM_CC_PolicyCpHash (TPM_CC)(0x0000016E)
#endif
#if CC_PolicyLocality
#define TPM_CC_PolicyLocality (TPM_CC)(0x0000016F)
#endif
#if CC_PolicyNameHash
#define TPM_CC_PolicyNameHash (TPM_CC)(0x00000170)
#endif
#if CC_PolicyOR
#define TPM_CC_PolicyOR (TPM_CC)(0x00000171)
#endif
#if CC_PolicyTicket
#define TPM_CC_PolicyTicket (TPM_CC)(0x00000172)
#endif
#if CC_ReadPublic
#define TPM_CC_ReadPublic (TPM_CC)(0x00000173)
#endif
#if CC_RSA_Encrypt
#define TPM_CC_RSA_Encrypt (TPM_CC)(0x00000174)
#endif
#if CC_StartAuthSession
758 #define TPM_CC_StartAuthSession             (TPM_CC)(0x00000176)
759 #endif
760 #if   CC_VerifySignature
761 #define TPM_CC_VerifySignature              (TPM_CC)(0x00000177)
762 #endif
763 #if   CC_ECC_Parameters
764 #define TPM_CC_ECC_Parameters               (TPM_CC)(0x00000178)
765 #endif
766 #if   CC_FirmwareRead
767 #define TPM_CC_FirmwareRead                 (TPM_CC)(0x00000179)
768 #endif
769 #if   CC_GetCapability
770 #define TPM_CC_GetCapability                (TPM_CC)(0x0000017A)
771 #endif
772 #if   CC_GetRandom
773 #define TPM_CC_GetRandom                    (TPM_CC)(0x0000017B)
774 #endif
775 #if   CC_GetTestResult
776 #define TPM_CC_GetTestResult                (TPM_CC)(0x0000017C)
777 #endif
778 #if   CC_Hash
779 #define TPM_CC_Hash                         (TPM_CC)(0x0000017D)
780 #endif
781 #if   CC_PCR_Read
782 #define TPM_CC_PCR_Read                     (TPM_CC)(0x0000017E)
783 #endif
784 #if   CC_PolicyPCR
785 #define TPM_CC_PolicyPCR                    (TPM_CC)(0x0000017F)
786 #endif
787 #if   CC_PolicyRestart
788 #define TPM_CC_PolicyRestart                (TPM_CC)(0x00000180)
789 #endif
790 #if   CC_ReadClock
791 #define TPM_CC_ReadClock                    (TPM_CC)(0x00000181)
792 #endif
793 #if   CC_PCR_Extend
794 #define TPM_CC_PCR_Extend                   (TPM_CC)(0x00000182)
795 #endif
796 #if   CC_PCR_SetAuthValue
797 #define TPM_CC_PCR_SetAuthValue             (TPM_CC)(0x00000183)
798 #endif
799 #if   CC_NV_Certify
800 #define TPM_CC_NV_Certify                  (TPM_CC)(0x00000184)
801 #endif
802 #if   CC_EventSequenceComplete
803 #define TPM_CC_EventSequenceComplete        (TPM_CC)(0x00000185)
804 #endif
805 #if   CC_HashSequenceStart
806 #define TPM_CC_HashSequenceStart            (TPM_CC)(0x00000186)
807 #endif
808 #if   CC_PolicyPhysicalPresence
809 #define TPM_CC_PolicyPhysicalPresence       (TPM_CC)(0x00000187)
810 #endif
811 #if   CC_PolicyDuplicationSelect
812 #define TPM_CC_PolicyDuplicationSelect      (TPM_CC)(0x00000188)
813 #endif
814 #if   CC_PolicyGetDigest
815 #define TPM_CC_PolicyGetDigest             (TPM_CC)(0x00000189)
816 #endif
817 #if   CC_TestParms
818 #define TPM_CC_TestParms                   (TPM_CC)(0x0000018A)
819 #endif
820 #if   CC_Commit
821 #define TPM_CC_Commit                     (TPM_CC)(0x0000018B)
822 #endif
823 #if   CC_PolicyPassword

#define TPM_CC_PolicyPassword (TPM_CC)(0x0000018C)

#ifdef CC_ZGen_2Phase
#define TPM_CC_ZGen_2Phase (TPM_CC)(0x0000018D)
#endif

#ifdef CC_EC_Ephemeral
#define TPM_CC_EC_Ephemeral (TPM_CC)(0x0000018E)
#endif

#ifdef CC_PolicyNvWritten
#define TPM_CC_PolicyNvWritten (TPM_CC)(0x0000018F)
#endif

#ifdef CC_PolicyTemplate
#define TPM_CC_PolicyTemplate (TPM_CC)(0x00000190)
#endif

#ifdef CC_CreateLoaded
#define TPM_CC_CreateLoaded (TPM_CC)(0x00000191)
#endif

#ifdef CC_PolicyAuthorizeNV
#define TPM_CC_PolicyAuthorizeNV (TPM_CC)(0x00000192)
#endif

#ifdef CC_EncryptDecrypt2
#define TPM_CC_EncryptDecrypt2 (TPM_CC)(0x00000193)
#endif

#ifdef CC_AC_GetCapability
#define TPM_CC_AC_GetCapability (TPM_CC)(0x00000194)
#endif

#ifdef CC_AC_Send
#define TPM_CC_AC_Send (TPM_CC)(0x00000195)
#endif

#ifdef CC_Policy_AC_SendSelect
#define TPM_CC_Policy_AC_SendSelect (TPM_CC)(0x00000196)
#endif

#define CC_VEND 0x20000000

#ifdef CC_Vendor_TCG_Test
#define TPM_CC_Vendor_TCG_Test (TPM_CC)(0x20000000)
#endif

#define TPM_CC_FIRST 0x0000011F
#define TPM_CC_LAST 0x00000196

#ifdef COMPRESSED_LISTS
#define ADD_FILL 0
#else
#define ADD_FILL 1
#endif

Size the array of library commands based on whether or not the array is packed (only defined commands) or dense (having entries for unimplemented commands)

#define LIBRARY_COMMAND_ARRAY_SIZE (0 |
+ (ADD_FILL || CC_NV.UndefineSpaceSpecial) /* 0x0000011F */ |
+ (ADD_FILL || CC_EvictControl) /* 0x00000120 */ |
+ (ADD_FILL || CC_HierarchyControl) /* 0x00000121 */ |
+ (ADD_FILL || CC_NV.UndefineSpace) /* 0x00000122 */ |
+ ADD_FILL /* 0x00000123 */ |
+ (ADD_FILL || CC_ChangeEPS) /* 0x00000124 */ |
+ (ADD_FILL || CC_ChangePPS) /* 0x00000125 */ |
+ (ADD_FILL || CC_Clear) /* 0x00000126 */ |
+ (ADD_FILL || CC_ClearControl) /* 0x00000127 */ |
+ (ADD_FILL || CC_HierarchyChangeAuth) /* 0x00000128 */ |
+ (ADD_FILL || CC_NV.UndefineSpace) /* 0x00000129 */ |
+ (ADD_FILL || CC_NV.UndefineSpaceSpecial) /* 0x0000012A */ |
+ (ADD_FILL || CC_PCR_Allocate) /* 0x0000012B */ |
+ (ADD_FILL || CC_PCR_SetAuthPolicy) /* 0x0000012C */ |)
Define the 2B structure that would hold any hash block
TPM2B_TYPE(MAX_HASH_BLOCK, MAX_HASH_BLOCK_SIZE);

Following typedef is for some old code

typedef TPM2B_MAX_HASH_BLOCK TPM2B_HASH_BLOCK;

Additional symmetric constants

ifndef ALG_AES
#define ALG_AES NO
#endif
ifndef MAX_AES_KEY_BITS
#define MAX_AES_KEY_BITS 0
#endif
ifndef MAX_AES_BLOCK_SIZE_BYTES
#define MAX_AES_BLOCK_SIZE_BYTES 0
#endif
ifndef ALG_CAMELLIA
#define ALG_CAMELLIA NO
#endif
ifndef MAX_CAMELLIA_KEY_BITS
#define MAX_CAMELLIA_KEY_BITS 0
#endif
ifndef MAX_CAMELLIA_BLOCK_SIZE_BYTES
#define MAX_CAMELLIA_BLOCK_SIZE_BYTES 0
#endif
ifndef ALG_SM4
#define ALG_SM4 NO
#endif
ifndef MAX_SM4_KEY_BITS
#define MAX_SM4_KEY_BITS 0
#endif
ifndef MAX_SM4_BLOCK_SIZE_BYTES
#define MAX_SM4_BLOCK_SIZE_BYTES 0
#endif
ifndef ALG_TDES
#define ALG_TDES NO
#endif
ifndef MAX_TDES_KEY_BITS
#define MAX_TDES_KEY_BITS 0
#endif
ifndef MAX_TDES_BLOCK_SIZE_BYTES
#define MAX_TDES_BLOCK_SIZE_BYTES 0
#endif

#define MAX_SYM_KEY_BITS (MAX(ALG_AES * MAX_AES_KEY_BITS, 
                         MAX(ALG_CAMELLIA * MAX_CAMELLIA_KEY_BITS, 
                         MAX(ALG_SM4 * MAX_SM4_KEY_BITS, 
                         MAX(ALG_TDES * MAX_TDES_KEY_BITS, 
                         0)))))
#define MAX_SYM_KEY_BYTES ((MAX_SYM_KEY_BITS + 7) / 8)
#define MAX_SYM_BLOCK_SIZE (MAX(ALG_AES * MAX_AES_BLOCK_SIZE_BYTES, 
                           MAX(ALG_CAMELLIA * MAX_CAMELLIA_BLOCK_SIZE_BYTES, 
                           MAX(ALG_SM4 * MAX_SM4_BLOCK_SIZE_BYTES, 
                           MAX(ALG_TDES * MAX_TDES_BLOCK_SIZE_BYTES, 
                           0))))

if MAX_SYM_KEY_BITS == 0 || MAX_SYM_BLOCK_SIZE == 0 
    error Bad size for MAX_SYM_KEY_BITS or MAX_SYM_BLOCK_SIZE
#endif // _IMPLEMENTATION_H_
Annex B
(informative)
Library-Specific

B.1 Introduction

This clause contains the files that are specific to a cryptographic library used by the TPM code. Three categories are defined for cryptographic functions:

1) big number math (asymmetric cryptography),
2) symmetric ciphers, and
3) hash functions.

The code is structured to make it possible to use different libraries for different categories. For example, one might choose to use OpenSSL for its math library, but use a different library for hashing and symmetric cryptography. Since OpenSSL supports all three categories, it might be more typical to combine libraries of specific functions; that is, one library might only contain block ciphers while another supports big number math.
B.2 OpenSSL-Specific Files

B.2.1. Introduction

The following files are specific to a port that uses the OpenSSL library for cryptographic functions.

B.2.2. Header Files

B.2.2.1. TpmToOsslHash.h

B.2.2.1.1. Introduction

This header file is used to splice the OpenSSL() hash code into the TPM code.

```c
#ifndef TPM_TO_OSSL_HASH_H
#define TPM_TO_OSSL_HASH_H

#if HASH_LIB == OSSL
#include <openssl/evp.h>
#include <openssl/sha.h>
#include <openssl/ossl_typ.h>
#endif

#define tpmHashStateSHA1_t        SHA_CTX
#define tpmHashStateSHA256_t      SHA256_CTX
#define tpmHashStateSHA384_t      SHA512_CTX
#define tpmHashStateSHA512_t      SHA512_CTX

#if ALG_SM3_256
#error "The version of OpenSSL used by this code does not support SM3"
#endif
```

B.2.2.1.2. Links to the OpenSSL HASH code

Redefine the internal name used for each of the hash state structures to the name used by the library. These defines need to be known in all parts of the TPM so that the structure sizes can be properly computed when needed.

```c
#define tpmHashStateSHA1_t        SHA_CTX
#define tpmHashStateSHA256_t      SHA256_CTX
#define tpmHashStateSHA384_t      SHA512_CTX
#define tpmHashStateSHA512_t      SHA512_CTX

#if ALG_SM3_256
#error "The version of OpenSSL used by this code does not support SM3"
#endif
```

The defines below are only needed when compiling CryptHash.c or CryptSmac.c. This isolation is primarily to avoid name space collision. However, if there is a real collision, it will likely show up when the linker tries to put things together.

```c
#ifdef _CRYPT_HASH_C_
typedef BYTE          *PBYTE;
typedef const BYTE    *PCBYTE;
#endif
```

Define the interface between CryptHash.c to the functions provided by the library. For each method, define the calling parameters of the method and then define how the method is invoked in CryptHash.c.

All hashes are required to have the same calling sequence. If they don't, create a simple adaptation function that converts from the standard form of the call to the form used by the specific hash (and then send a nasty letter to the person who wrote the hash function for the library).

The macro that calls the method also defines how the parameters get swizzled between the default form (in CryptHash.c) and the library form.

Initialize the hash context

```c
#define HASH_START_METHOD_DEF void (HASH_START_METHOD)(PANY_HASH_STATE state)
#define HASH_START(hashState) 
```
Add data to the hash

```c
#define HASH_DATA_METHOD_DEF
void (HASH_DATA_METHOD)(PANY_HASH_STATE state,
PCBYTE buffer,
size_t size)
#define HASH_DATA(hashState, dInSize, dIn)
((hashState)->def->method.start)(&(hashState)->state);
```

Finalize the hash and get the digest

```c
#define HASH_END_METHOD_DEF
void (HASH_END_METHOD)(BYTE *buffer, PANY_HASH_STATE state)
#define HASH_END(hashState, buffer)
((hashState)->def->method.end)(buffer, &(hashState)->state)
```

Copy the hash context

```c
#define HASH_STATE_COPY_METHOD_DEF
void (HASH_STATE_COPY_METHOD)(PANY_HASH_STATE to,
PCANY_HASH_STATE from,
size_t size)
#define HASH_STATE_COPY(hashStateOut, hashStateIn)
((hashStateIn)->def->method.copy)(&(hashStateOut)->state,
&(hashStateOut)->state,
(hashStateIn)->def->contextSize)
```

Copy (with reformatting when necessary) an internal hash structure to an external blob

```c
#define HASH_STATE_EXPORT_METHOD_DEF
void (HASH_STATE_EXPORT_METHOD)(BYTE *to,
PCANY_HASH_STATE from,
size_t size)
#define HASH_STATE_EXPORT(to, hashStateFrom)
((hashStateFrom)->def->method.copyOut)
&(((BYTE *)(to))[offsetof(HASH_STATE, state)],
&(hashStateFrom)->state,
(hashStateFrom)->def->contextSize)
```

Copy from an external blob to an internal formate (with reformatting when necessary

```c
#define HASH_STATE_IMPORT_METHOD_DEF
void (HASH_STATE_IMPORT_METHOD)(PANY_HASH_STATE to,
const BYTE *from,
size_t size)
#define HASH_STATE_IMPORT(hashStateTo, from)
((hashStateTo)->def->method.copyIn)
&((((const BYTE *)(from))[offsetof(HASH_STATE, state)]),
(hashStateTo)->state,
(hashStateTo)->def->contextSize)
```

Function aliases. The code in CryptHash.c uses the internal designation for the functions. These need to be translated to the function names of the library.
<table>
<thead>
<tr>
<th>Internal</th>
<th>External</th>
</tr>
</thead>
<tbody>
<tr>
<td>Designation</td>
<td>Designation</td>
</tr>
<tr>
<td>#define tpmHashStart_SHA1           SHA1_Init // external name of the initialization method</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashData_SHA1            SHA1_Update</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashEnd_SHA1             SHA1_Final</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateCopy_SHA1       memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateExport_SHA1     memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateImport_SHA1     memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStart_SHA256         SHA256_Init</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashData_SHA256          SHA256_Update</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashEnd_SHA256           SHA256_Final</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateCopy_SHA256     memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateExport_SHA256   memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateImport_SHA256   memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStart_SHA384         SHA384_Init</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashData_SHA384          SHA384_Update</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashEnd_SHA384           SHA384_Final</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateCopy_SHA384     memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateExport_SHA384   memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateImport_SHA384   memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStart_SHA512         SHA512_Init</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashData_SHA512          SHA512_Update</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashEnd_SHA512           SHA512_Final</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateCopy_SHA512     memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateExport_SHA512   memcpy</td>
<td></td>
</tr>
<tr>
<td>#define tpmHashStateImport_SHA512   memcpy</td>
<td></td>
</tr>
<tr>
<td>#define LibHashInit()</td>
<td></td>
</tr>
<tr>
<td>This definition would change if there were something to report</td>
<td></td>
</tr>
<tr>
<td>#define HashLibSimulationEnd()</td>
<td></td>
</tr>
<tr>
<td>#endif // HASH_LIB == OSSL</td>
<td></td>
</tr>
<tr>
<td>#endif // HASH_LIB == OSSL</td>
<td></td>
</tr>
<tr>
<td>#endif // HASH_LIB == OSSL</td>
<td></td>
</tr>
<tr>
<td>#endif // HASH_LIB == OSSL</td>
<td></td>
</tr>
</tbody>
</table>
B.2.2.2. TpmToOsslMath.h

B.2.2.2.1. Introduction

This file contains the structure definitions used for ECC in the LibTopCrypt() version of the code. These definitions would change, based on the library. The ECC-related structures that cross the TPM interface are defined in TpmTypes.h

```c
#ifndef _TPM_TO_OSSL_MATH_H
#define _TPM_TO_OSSL_MATH_H
#if MATH_LIB == OSSL
#include <openssl/evp.h>
#include <openssl/ec.h>
#if OPENSSL_VERSION_NUMBER >= 0x10100000L
#include <openssl/bn_lcl.h>
#endif
#include <openssl/bn.h>
#endif
#endif
```

B.2.2.2.2. Macros and Defines

Make sure that the library is using the correct size for a crypt word

```c
#if defined THIRTY_TWO_BIT && (RADIX_BITS != 32) \n|| (defined SIXTY_FOUR_BIT_LONG || defined SIXTY_FOUR_BIT) \n&& (RADIX_BITS != 64))
#  error "Ossl library is using different radix"
#endif
```

Allocate a local BIGNUM value. For the allocation, a bigNum structure is created as is a local BIGNUM. The bigNum is initialized and then the BIGNUM is set to reference the local value.

```c
#define BIG_VAR(name, bits) \nBN_VAR(name##Bn, (bits)); \nBIGNUM _##name; \nBIGNUM *name = BigInitialized(&_##name, \nBNInit(name##Bn, \nBYTES_TO_CRYPT_WORDS(sizeof(_##name##Bn.d))))
```

Allocate a BIGNUM and initialize with the values in a bigNum initializer

```c
#define BIG_INITIALIZED(name, initializer) \nOSSL_CURVE_DATA _##name; \nbigCurve name =  BnCurveInitialize(&_##name, initializer)
```

```c
typedef struct
{
    const ECC_CURVE_DATA  *C;  // the TPM curve values
    EC_GROUP               *G;  // group parameters
    BN_CTX                 *CTX; // the context for the math (this might not be
                                // the context in which the curve was created>;
} OSSL_CURVE_DATA;

typedef OSSL_CURVE_DATA *bigCurve;
#define AccessCurveData(E) ((E)->C)
#define CURVE_INITIALIZED(name, initializer) \nbigCurve name =  BnCurveInitialize(&_##name, initializer)
```

```c
#include "TpmToOsslSupport_fp.h"
#define CURVE_FREE(E) \nif(E != NULL) \n{ \n    if(E->G != NULL) \n        EC_GROUP_free(E->G);
```
OsslContextLeave(E-CTX);

#define OSSL_ENTER() BN_CTX *CTX = OsslContextEnter()
#define OSSL_LEAVE() OsslContextLeave(CTX)

This definition would change if there were something to report

#define MathLibSimulationEnd()
#endif // MATH_LIB == OSSL
#endif
B.2.2.3.  TpmToOsslSym.h

B.2.2.3.1.  Introduction

This header file is used to *splice* the OpenSSL() library into the TPM code.

The support required of a library are a hash module, a block cipher module and portions of a big number library.

```
#ifndef _TPM_TO_OSSL_SYM_H_
#define _TPM_TO_OSSL_SYM_H_
#if SYM_LIB == OSS
#include <openssl/aes.h>
#include <openssl/des.h>
#include <openssl/bn.h>
#include <openssl/ossl_typ.h>
```

B.2.2.3.2.  Links to the OpenSSL AES code

```
#if ALG_SM4
#error "SM4 is not available"
#endif
#if ALG_CAMELLIA
#error "Camellia is not available"
#endif
```

Define the order of parameters to the library functions that do block encryption and decryption.

```
typedef void (*TpmCryptSetSymKeyCall_t)(
    const BYTE *in,
    BYTE        *out,
    void        *keySchedule
);
```

The Crypt functions that call the block encryption function use the parameters in the order:

a)  *keySchedule

b)  in buffer

c)  out buffer Since open SSL uses the order in *encryptoCall_t* above, need to swizzle the values to the order required by the library.

```
#define SWIZZLE(keySchedule, in, out)
    (const BYTE *)(in), (BYTE *)(out), (void *)(keySchedule)
```

Macros to set up the encryption/decryption key schedules

AES:
```
#define TpmCryptSetEncryptKeyAES(key, keySizeInBits, schedule) ...
    AES_set_encrypt_key((key), (keySizeInBits), (tpmKeyScheduleAES *)(schedule))
```

TDES:
```
#define TpmCryptSetEncryptKeyTDES(key, keySizeInBits, schedule) ...
    TDES_set_encrypt_key((key), (keySizeInBits), (tpmKeyScheduleTDES *)(schedule))
```
Macros to alias encryption calls to specific algorithms. This should be used sparingly. Currently, only used by CryptRand.c

When using these calls, to call the AES block encryption code, the caller should use:
\texttt{TpmCryptEncryptAES(SWIZZLE(keySchedule, in, out))};

\begin{verbatim}
29  #define TpmCryptEncryptAES   AES_encrypt
30  #define TpmCryptDecryptAES  AES_decrypt
31  #define tpmKeyScheduleAES   AES_KEY
32  #define TpmCryptEncryptTDES TDES_encrypt
33  #define TpmCryptDecryptTDES TDES_decrypt
34  #define tpmKeyScheduleTDES  DES_key_schedule
35  typedef union tpmCryptKeySchedule_t tpmCryptKeySchedule_t;
36  #if ALG_TDES
37  #include "TpmToOsslDesSupport_fp.h"
38  #endif
\end{verbatim}

This definition would change if there were something to report

\begin{verbatim}
39  #define SymLibSimulationEnd()
40  #endif // SYM_LIB == OSSL
41  #endif // _TPM_TO_OSSL_SYM_H_
\end{verbatim}
B.2.3. Source Files

B.2.3.1. TpmToOsslDesSupport.c

B.2.3.1.1. Introduction

The functions in this file are used for initialization of the interface to the OpenSSL() library.

B.2.3.1.2. Defines and Includes

```
#include "Tpm.h"
#ifndef SYM_LIB == OSSL && defined TPM_ALG_TDES

B.2.3.1.3. Functions

B.2.3.1.3.1. TDES_set_encrpyt_key()

This function makes creation of a TDES key look like the creation of a key for any of the other OpenSSL() block ciphers. It will create three key schedules, one for each of the DES keys. If there are only two keys, then the third schedule is a copy of the first.

void TDES_set_encrypt_key(
    const BYTE *key,
    UINT16 keySizeInBits,
    tpmKeypScheduleTDES *keySchedule)
{
    DES_set_key_unchecked((const DES_cblock *)key, &keySchedule[0]);
    DES_set_key_unchecked((const DES_cblock *)&key[8], &keySchedule[1]);
    // If is two-key, copy the schedule for K1 into K3, otherwise, compute the
    // the schedule for K3
    if(keySizeInBits == 128)
        keySchedule[2] = keySchedule[0];
    else
        DES_set_key_unchecked((const DES_cblock *)&key[16],
                              &keySchedule[2]);
}

B.2.3.1.3.2. TDES_encrypt()

The TPM code uses one key schedule. For TDES, the schedule contains three schedules. OpenSSL() wants the schedules referenced separately. This function does that.

void TDES_encrypt(
    const BYTE *in,
    BYTE *out,
    tpmKeypScheduleTDES *ks)
{
    DES_ecb3_encrypt((const DES_cblock *)in, (DES_cblock *)out,
                     &ks[0], &ks[1], &ks[2],
                     DES_ENCRYPT);
}
B.2.3.1.3.3. TDES_decrypt()

As with TDES_encrypt() this function bridges between the TPM single schedule model and the OpenSSL() three schedule model.

```c
void TDES_decrypt(
    const BYTE *in,
    BYTE *out,
    tpmKeyScheduleTDES *ks
)
{
    DES_ecb3_encrypt((const_DES_cblock *)in, (DES_cblock *)out,
                     ks[0], ks[1], ks[2],
                     DES_DECRYPT);
}
#endif // SYM_LIB == OSSL
```
B.2.3.2. TpmToOsslMath.c

B.2.3.2.1. Introduction

This file contains the math functions that are not implemented in the BnMath() library (yet). These math functions will call the OpenSSL() library to execute the operations. There is a difference between the internal format and the OpenSSL() format. To call the OpenSSL() function, a BIGNUM structure is created for each passed variable. The sizes in the bignum_t are copied and the d pointer in the BIGNUM is set to point to the d parameter of the bignum_t. On return, SetSizeOsslToTpm() is used for each returned variable to make sure that the pointers are not changed. The size of the returned BIGGNUM is copied to bignum_t.

B.2.3.2.2. Includes and Defines

```c
#include "Tpm.h"
#if MATH_LIB == OSSL
#include "TpmToOsslMath_fp.h"
#endif
```

B.2.3.2.3. Functions

B.2.3.2.3.1. OsslToTpmBn()

This function converts an OpenSSL() BIGNUM to a TPM bignum. In this implementation it is assumed that OpenSSL() used the same format for a big number as does the TPM -- an array of native-endian words in little-endian order.

If the array allocated for the OpenSSL() BIGNUM is not the space within the TPM bignum, then the data is copied. Otherwise, just the size field of the BIGNUM is copied.

```c
void OsslToTpmBn(
    bigNum bn,
    BIGNUM *osslBn
)
{
    if(bn != NULL)
    {
        if((crypt_uword_t *)osslBn->d != bn->d)
        {
            int i;
            pAssert((unsigned)osslBn->top <= BnGetAllocated(bn));
            for(i = 0; i < osslBn->top; i++)
                bn->d[i] = osslBn->d[i];
            BnSetTop(bn, osslBn->top);
        }
    }
}
```

B.2.3.2.3.2. BigInitialized()

This function initializes an OSSL BIGNUM from a TPM bignum.

```c
BIGNUM * BigInitialized(
    BIGNUM *toInit,
    bigConst initializer
)
{
```

---
if (toInit == NULL || initializer == NULL)
    return NULL;

    toInit->d = (BN_ULONG *)&initializer->d[0];
    toInit->dmax = initializer->allocated;
    toInit->top = initializer->size;
    toInit->neg = 0;
    toInit->flags = 0;
    return toInit;
}
#endif OSSL_DEBUG
#define BIGNUM_PRINT(label, bn, eol)
#define DEBUG_PRINT(x)
#else
#define DEBUG_PRINT(x)   printf("%s", x)
#define BIGNUM_PRINT(label, bn, eol) BIGNUM_print((label), (bn), (eol))
static
void BIGNUM_print(
    const char  *label,
    const BIGNUM  *a,
    BOOL             eol
)
{
     BN_ULONG        *d;
     int              i;
     int              notZero = FALSE;

     if(label != NULL)
         printf("%s", label);
     if(a == NULL)
     {
         printf("NULL");
         goto done;
     }
     if (a->neg)
         printf("-");
     for(i = a->top, d = &a->d[i - 1]; i > 0; i--)
     {
         int             j;
         BN_ULONG        l = *d--;
         for(j = BN_BITS2 - 8; j >= 0; j -= 8)
         {
             BYTE           b = (BYTE)((l >> j) & 0xFF);
             notZero = notZero || (b != 0);
             if(notZero)
                 printf("%02x", b);
         }
         if(!notZero)
             printf("0");
     }
     done:
     if(eol)
         printf("\n");
     return;
}
#endif LIBRARY_COMPATIBILITY_CHECK
#define MathLibraryCompatibilityCheck(
    void
)
{
    OSSL_ENTER();
    BIGNUM          *osslTemp = BN_CTX_get(CTX);
    BN_VAR(tpmTemp, 64 * 8); // allocate some space for a test value
crypt_uword_t i;
    TPM2B_TYPE(TEST, 16);
B.2.3.2.3.3. BnModMult()

Does multiply and divide returning the remainder of the divide.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation</td>
</tr>
</tbody>
</table>

B.2.3.2.3.4. BnMult()

Multiplies two numbers
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation</td>
</tr>
</tbody>
</table>

```
LIB_EXPORT BOOL BnMult(
    bigNum result,
    bigConst multiplicand,
    bigConst multiplier
) {
    OSSL_ENTER();
    BN_VAR(temp, (LARGEST_NUMBER_BITS * 2));
    BIG_INITIALIZED(bnTemp, temp);
    BIG_INITIALIZED(bnA, multiplicand);
    BIG_INITIALIZED(bnB, multiplier);
    BOOL OK;
    pAssert(result->allocated >=
        (BITS_TO_CRYPT_WORDS(BnSizeInBits(multiplicand)
          + BnSizeInBits(multiplier))));
    OK = BN_mul(bnTemp, bnA, bnB, CTX);
    if (OK) {
        OsslToTpmBn(temp, bnTemp);
        BnCopy(result, temp);
    }
    OSSL_LEAVE();
    return OK;
}
```

B.2.3.2.3.5. BnDiv()

This function divides two bigNum values. The function returns FALSE if there is an error in the operation.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation</td>
</tr>
</tbody>
</table>

```
LIB_EXPORT BOOL BnDiv(
    bigNum quotient,
    bigNum remainder,
    bigConst dividend,
    bigConst divisor
) {
    OSSL_ENTER();
    BIG_INITIALIZED(bnQ, quotient);
    BIG_INITIALIZED(bnR, remainder);
    BIG_INITIALIZED(bnDend, dividend);
    BIG_INITIALIZED(bnSor, divisor);
    BOOL OK;
    pAssert(!BnEqualZero(divisor));
    if (BnGetSize(dividend) < BnGetSize(divisor))
        if (quotient)
            BnSetWord(quotient, 0);
        if (remainder)
            BnCopy(remainder, dividend);
    OK = TRUE;
```
182     }
183     else
184     {
185         pAssert((quotient == NULL)
186         || (quotient->allocated >= (unsigned)(dividend->size
187             - divisor->size)));
188         pAssert((remainder == NULL)
189         || (remainder->allocated >= divisor->size));
190         OK = BN_div(bnQ, bnR, bnDend, bnSor, CTX);
191         if(OK)
192         {
193             OsslToTpmBn(quotient, bnQ);
194             OsslToTpmBn(remainder, bnR);
195         }
196         DEBUG_PRINT("In BnDiv:\n");
197         BIGNUM_PRINT("    bnDividend: ", bnDend, TRUE);
198         BIGNUM_PRINT("    bnDivisor: ", bnSor, TRUE);
199         BIGNUM_PRINT("    bnQuotient: ", bnQ, TRUE);
200         BIGNUM_PRINT("    bnRemainder: ", bnR, TRUE);
201         OSSL_LEAVE();
202         return OK;
203     }
204 #if ALG_RSA

B.2.3.2.3.6. BnGcd()

Get the greatest common divisor of two numbers

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation</td>
</tr>
</tbody>
</table>

206 LIB_EXPORT BOOL
207 BnGcd(
208     bigNum      gcd,       // OUT: the common divisor
209     bigConst    number1,  // IN:
210     bigConst    number2  // IN:
211     )
212 {
213     OSSL_ENTER();
214     BIG_INITIALIZED(bnGcd, gcd);
215     BIG_INITIALIZED(bn1, number1);
216     BIG_INITIALIZED(bn2, number2);
217     BOOL OK;
218     pAssert(gcd != NULL);
219     OK = BN_gcd(bnGcd, bn1, bn2, CTX);
220     if(OK)
221     {
222         OsslToTpmBn(gcd, bnGcd);
223         gcd->size = bnGcd->top;
224     }
225     OSSL_LEAVE();
226     return OK;
227 }

B.2.3.2.3.7. BnModExp()

Do modular exponentiation using bigNum values. The conversion from a bignum_t to a bigNum is trivial as they are based on the same structure
### B.2.3.2.3.8. BnModInverse()

Modular multiplicative inverse

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT BOOL BnModInverse(
    bigNum result,  // OUT: the result
    bigConst number,  // IN: number to exponentiate
    bigConst exponent,  // IN:
    bigConst modulus  // IN:
)
{
    OSSL_ENTER();
    BIG_INITIALIZED(bnResult, result);
    BIG_INITIALIZED(bnN, number);
    BIG_INITIALIZED(bnE, exponent);
    BIG_INITIALIZED(bnM, modulus);
    BOOL OK;

    OK = BN_mod_inverse(bnResult, bnN, bnE, bnM, CTX) != NULL;
    if(OK)
    {
        OsslToTpmBn(result, bnResult);
    }
    OSSL_LEAVE();
    return OK;
}
```

```c
#define // ALG_RSA
```

```c
#elif ALG_ECC
```
B.2.3.2.3.9. PointFromOssl()

Function to copy the point result from an OSSL function to a \textit{bigNum}

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation</td>
</tr>
</tbody>
</table>

```c
static BOOL PointFromOssl(
    bigPoint         pOut,  // OUT: resulting point
    EC_POINT        *pIn,   // IN: the point to return
    bigCurve         E     // IN: the curve
)
{
    BIGNUM         *x = NULL;
    BIGNUM         *y = NULL;
    BOOL            OK;

    BN_CTX_start(E->CTX);
    x = BN_CTX_get(E->CTX);
    y = BN_CTX_get(E->CTX);
    if(y == NULL)
        FAIL(FATAL_ERROR_ALLOCATION);
    if(this returns false, then the point is at infinity
    OK = EC_POINT_get_affine_coordinates_GFp(E->G, pIn, x, y, E->CTX);
    if(OK)
        { OsslToTpmBn(pOut->x, x);
          OsslToTpmBn(pOut->y, y);
          BnSetWord(pOut->z, 1);
        }
    else
        BnSetWord(pOut->z, 0);
    BN_CTX_end(E->CTX);
    return OK;
}
```

B.2.3.2.3.10. EcPointInitialized()

Allocate and initialize a point.

```c
static EC_POINT * EcPointInitialized(
    pointConst          initializer,
    bigCurve            E)
{
    BIG_INITIALIZED(bnX, (initializer != NULL) ? initializer->x : NULL);
    BIG_INITIALIZED(bnY, (initializer != NULL) ? initializer->y : NULL);
    EC_POINT            *P = (initializer != NULL && E != NULL)
        ? EC POINT new(E->G) : NULL;
    pAssert(E != NULL);
    if(P != NULL)
        { EC POINT_set_affine_coordinates_GFp(E->G, P, bnX, bnY, E->CTX);
          return P;
        }
```
B.2.3.2.3.11. BnCurveInitialize()

This function initializes the OpenSSL() group definition. It is a fatal error if groupContext is not provided.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>the TPM_ECC_CURVE is not valid</td>
</tr>
<tr>
<td>non-NULL</td>
<td>points to a structure in groupContext</td>
</tr>
</tbody>
</table>

```c
bigCurve
BnCurveInitialize(E, TPM_ECC_CURVE curveId)
{
    EC_GROUP *group = NULL;
    EC_POINT *P = NULL;
    const ECC_CURVE_DATA *C = GetCurveData(curveId);
    BN_CTX *CTX = NULL;

    BIG_INITIALIZED(bnP, C != NULL ? C->prime : NULL);
    BIG_INITIALIZED(bnA, C != NULL ? C->a : NULL);
    BIG_INITIALIZED(bnB, C != NULL ? C->b : NULL);
    BIG_INITIALIZED(bnX, C != NULL ? C->base.x : NULL);
    BIG_INITIALIZED(bnY, C != NULL ? C->base.y : NULL);
    BIG_INITIALIZED(bnN, C != NULL ? C->order : NULL);
    BIG_INITIALIZED(bnH, C != NULL ? C->h : NULL);

    int OK = (C != NULL);
    OK = OK && ((CTX = OsslContextEnter()) != NULL);

    // initialize EC group, associate a generator point and initialize the point
    // from the parameter data
    // Create a group structure
    OK = OK && (group = EC_GROUP_new_curve_GFp(bnP, bnA, bnB, CTX)) != NULL;

    // Allocate a point in the group that will be used in setting the
    // generator. This is not needed after the generator is set.
    OK = OK && ((P = EC_POINT_new(group)) != NULL);

    // Need to use this in case Montgomery method is being used
    OK = OK && EC_POINT_set_affine_coordinates_GFp(group, P, bnX, bnY, CTX);

    // Now set the generator
    OK = OK && EC_GROUP_set_generator(group, P, bnN, bnH);

    if(P != NULL)
        EC_POINT_free(P);

    if(!OK && group != NULL)
        { EC_GROUP_free(group);
          group = NULL;
        }

    if(!OK && CTX != NULL)
        { OsslContextLeave(CTX);
          CTX = NULL;
        }

    if(E->G = group);
    E->CTX = CTX;
    E->C = C;
```
B.2.3.2.3.12. **BnEccModMult()**

This function does a point multiply of the form \( R = [d]S \)

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation; treat as result being point at infinity</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT BOOL
BnEccModMult(
    bigPoint R,  // OUT: computed point
    pointConst S, // IN: point to multiply by 'd' (optional)
    bigConst d,  // IN: scalar for [d]S
    bigCurve E   // IN: curve
) {
    EC_POINT *pR = EC_POINT_new(E->G);
    EC_POINT *pS = EcPointInitialized(S, E);
    BIG_INITIALIZED(bnD, d);

    if (S == NULL)
        EC_POINT_mul(E->G, pR, bnD, NULL, NULL, E->CTX);
    else
        EC_POINT_mul(E->G, pR, NULL, pS, bnD, E->CTX);
    PointFromOssl(R, pR, E);
    return !BnEqualZero(R->z);
}
```

B.2.3.2.3.13. **BnEccModMult2()**

This function does a point multiply of the form \( R = [d]G + [u]Q \)

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation; treat as result being point at infinity</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT BOOL
BnEccModMult2(
    bigPoint R,  // OUT: computed point
    pointConst S, // IN: optional point
    bigConst d,  // IN: scalar for [d]S or [d]G
    pointConst Q, // IN: second point
    bigConst u,  // IN: second scalar
    bigCurve E   // IN: curve
) {
    EC_POINT *pR = EC_POINT_new(E->G);
    EC_POINT *pS = EcPointInitialized(S, E);
    BIG_INITIALIZED(bnD, d);

    if (S == NULL || S == (pointConst)&(AccessCurveData(E)->base))
        BIG_INITIALIZED(bnU, u);
```
```c
414     EC_POINT_mul(E->G, pR, bnD, pQ, bnU, E->CTX);
415     else
416     {
417         const EC_POINT    *points[2];
418         const BIGNUM       *scalars[2];
419         points[0] = pS;
420         points[1] = pQ;
421         scalars[0] = bnD;
422         scalars[1] = bnU;
423         EC_POINTs_mul(E->G, pR, NULL, 2, points, scalars, E->CTX);
424     }
425     PointFromOssl(R, pR, E);
426     EC_POINT_free(pR);
427     EC_POINT_free(pS);
428     EC_POINT_free(pQ);
429     return !BnEqualZero(R->z);
430 }

B.2.3.2.4. BnEccAdd()

This function does addition of two points.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure in operation; treat as result being point at infinity</td>
</tr>
</tbody>
</table>

```
B.2.3.3. TpmToOsslSupport.c

B.2.3.3.1. Introduction

The functions in this file are used for initialization of the interface to the OpenSSL() library.

B.2.3.3.2. Defines and Includes

1 #include "Tpm.h"
2 #if MATH_LIB == OSSL

Used to pass the pointers to the correct sub-keys

3 typedef const BYTE *desKeyPointers[3];

B.2.3.3.2.1. SupportLibInit()

This does any initialization required by the support library.

4 LIB_EXPORT int
5 SupportLibInit(
6     void
7 )
8 {
9     #ifdef LIBRARY_COMPATIBILITY_CHECK
10         MathLibraryCompatibilityCheck();
11     #endif
12     return TRUE;
13 }

B.2.3.3.2.2. OsslContextEnter()

This function is used to initialize an OpenSSL() context at the start of a function that will call to an
OpenSSL() math function.

14 BN_CTX *
15 OsslContextEnter(
16     void
17 )
18 {
19     BN_CTX *context = BN_CTX_new();
20     if(context == NULL)
21         FAIL(FATAL_ERROR_ALLOCATION);
22     BN_CTX_start(context);
23     return context;
24 }

B.2.3.3.2.3. OsslContextLeave()

This is the companion function to OsslContextEnter().

25 void
26 OsslContextLeave(
27     BN_CTX *context
28 )
29 {
30     if(context != NULL)
31     {
32         BN_CTX_end(context);
33     }
34 }
BN_CTX_end(context);
BN_CTX_free(context);
}
#endif // MATH_LIB == OSSL


Annex C
(informative)
Simulation Environment

C.1 Introduction

These files are used to simulate some of the implementation-dependent hardware of a TPM. These files are provided to allow creation of a simulation environment for the TPM. These files are not expected to be part of a hardware TPM implementation.

C.2 Cancel.c

C.2.1 Description

This module simulates the cancel pins on the TPM.

C.2.2 Includes, Typedefs, Structures, and Defines

```
#include "PlatformData.h"
#include "Platform_fp.h"
```

C.2.3 Functions

C.2.3.1 _plat__IsCanceled()

Check if the cancel flag is set

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>if cancel flag is set</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>if cancel flag is not set</td>
</tr>
</tbody>
</table>

```
LIBEXPORT int
_plat__IsCanceled(
    void
)
{
    // return cancel flag
    return s_isCanceled;
}
```

C.2.3.2 _plat__SetCancel()

Set cancel flag.

```
LIBEXPORT void
_plat__SetCancel(
    void
)
{
    s_isCanceled = TRUE;
    return;
}
```
C.2.3.3. _plat__ClearCancel()

Clear cancel flag

19 LIB_EXPORT void
20 _plat__ClearCancel ( 21     void
22       )
23 {
24     s_isCanceled = FALSE;
25     return;
26 }
C.3 Clock.c

C.3.1 Description

This file contains the routines that are used by the simulator to mimic a hardware clock on a TPM.
In this implementation, all the time values are measured in millisecond. However, the precision of the clock functions may be implementation dependent.

C.3.2 Includes and Data Definitions

```
#include "PlatformData.h"
#include "Platform_fp.h"
#include "TpmFail_fp.h"
#include <assert.h>
```

C.3.3 Simulator Functions

C.3.3.1 Introduction

This set of functions is intended to be called by the simulator environment in order to simulate hardware events.

C.3.3.2 __plat__TimerReset()

This function sets current system clock time as t0 for counting TPM time. This function is called at a power on event to reset the clock. When the clock is reset, the indication that the clock was stopped is also set.

```
LIBEXPORT void
__plat__TimerReset()
{
    s_lastSystemTime = 0;
    s_tpmTime = 0;
    s_adjustRate = CLOCK_NOMINAL;
    s_timerReset = TRUE;
    s_timerStopped = TRUE;
    return;
}
```

C.3.3.3 __plat__TimerRestart()

This function should be called in order to simulate the restart of the timer should it be stopped while power is still applied.

```
LIBEXPORT void
__plat__TimerRestart()
{
    void
    }
    s_timerStopped = TRUE;
    return;
}
```
C.3.4. Functions Used by TPM

C.3.4.1. Introduction

These functions are called by the TPM code. They should be replaced by appropriated hardware functions.

```
#include <time.h>
clock_t debugTime;
```

C.3.4.2. _plat__RealTime()

This is another, probably futile, attempt to define a portable function that will return a 64-bit clock value that has mSec resolution.

```
uint64_t _plat__RealTime( void ) {
  clock64_t time;
  #ifdef _MSC_VER
  struct _timeb sysTime;
  // _ftime_s(&sysTime);
  time = (clock64_t)(sysTime.time) * 1000 + sysTime.millitm;
  // set the time back by one hour if daylight savings
  if(sysTime.dstflag)
    time -= 1000 * 60 * 60; // mSec/sec * sec/min * min/hour = ms/hour
  #else
  // hopefully, this will work with most UNIX systems
  struct timespec systime;
  // clock_gettime(CLOCK_MONOTONIC, &systime);
  time = (clock64_t)systime.tv_sec * 1000 + (systime.tv_nsec / 1000000);
  #endif
  return time;
}
```

C.3.4.3. _plat__TimerRead()

This function provides access to the tick timer of the platform. The TPM code uses this value to drive the TPM Clock.

The tick timer is supposed to run when power is applied to the device. This timer should not be reset by time events including _TPM_Init(). It should only be reset when TPM power is re-applied.

If the TPM is run in a protected environment, that environment may provide the tick time to the TPM as long as the time provided by the environment is not allowed to go backwards. If the time provided by the system can go backwards during a power discontinuity, then the _plat__Signal_PowerOn() should call _plat__TimerReset().

```
LIB_EXPORT uint64_t _plat__TimerRead( void ) {
  #ifdef HARDWARE_CLOCK
  #error "need a definition for reading the hardware clock"
  return HARDWARE_CLOCK
  ```
#else
    clock64_t         timeDiff;
    clock64_t         adjustedTimeDiff;
    clock64_t         timeNow;
    clock64_t         readjustedTimeDiff;

    // This produces a timeNow that is basically locked to the system clock.
    timeNow = _plat__RealTime();

    // if this hasn't been initialized, initialize it
    if(s_lastSystemTime == 0)
    {
        s_lastSystemTime = timeNow;
        debugTime = clock();
        s_lastReportedTime = 0;
        s_realTimePrevious = 0;
    }

    // The system time can bounce around and that's OK as long as we don't allow
    // time to go backwards. When the time does appear to go backwards, set
    // lastSystemTime to be the new value and then update the reported time.
    if(timeNow < s_lastReportedTime)
    {
        s_lastSystemTime = timeNow;
        s_lastReportedTime = s_lastReportedTime + timeNow - s_lastSystemTime;
        s_lastSystemTime = timeNow;
    }

    // The code above produces a timeNow that is similar to the value returned
    // by Clock(). The difference is that timeNow does not max out, and it is
    // at a ms. rate rather than at a CLOCKS_PER_SEC rate. The code below
    // uses that value and does the rate adjustment on the time value.
    // If there is no difference in time, then skip all the computations
    if(s_realTimePrevious >= timeNow)
    {
        return s_tpmTime;
    }

    // Compute the amount of time since the last update of the system clock
    timeDiff = timeNow - s_realTimePrevious;

    // Do the time rate adjustment and conversion from CLOCKS_PER_SEC to mSec
    adjustedTimeDiff = (timeDiff * CLOCK_NOMINAL) / ((uint64_t)s_adjustRate);

    // update the TPM time with the adjusted timeDiff
    s_tpmTime += (clock64_t)adjustedTimeDiff;

    // Might have some rounding error that would loose CLOCKS. See what is not
    // being used. As mentioned above, this could result in putting back more than
    // is taken out. Here, we are trying to recreate timeDiff.
    readjustedTimeDiff = (adjustedTimeDiff * (uint64_t)s_adjustRate )
    / CLOCK_NOMINAL;

    // adjusted is now converted back to being the amount we should advance the
    // previous sampled time. It should always be less than or equal to timeDiff.
    // That is, we could not have use more time than we started with.
    s_realTimePrevious = s_realTimePrevious + readjustedTimeDiff;

#ifdef DEBUGGING_TIME
    // Put this in so that TPM time will pass much faster than real time when
    // doing debug.
    // A value of 1000 for DEBUG_TIME_MULTIPLER will make each ms into a second
    // A good value might be 100
    return (s_tpmTime * DEBUG_TIME_MULTIPLIER);
#endif
#endif
return s_tpmTime;
#endif}
C.3.4.4. \_plat\_TimerWasReset()

This function is used to interrogate the flag indicating if the tick timer has been reset.

If the resetFlag parameter is SET, then the flag will be CLEAR before the function returns.

```c
LIB_EXPORT BOOL
plat_TimerWasReset(
    void
)
{
    BOOL         retVal = s_timerReset;
    s_timerReset = FALSE;
    return retVal;
}
```

C.3.4.5. \_plat\_TimerWasStopped()

This function is used to interrogate the flag indicating if the tick timer has been stopped. If so, this is typically a reason to roll the nonce.

This function will CLEAR the s_timerStopped flag before returning. This provides functionality that is similar to status register that is cleared when read. This is the model used here because it is the one that has the most impact on the TPM code as the flag can only be accessed by one entity in the TPM. Any other implementation of the hardware can be made to look like a read-once register.

```c
LIB_EXPORT BOOL
plat_TimerWasStopped(
    void
)
{
    BOOL         retVal = s_timerStopped;
    s_timerStopped = FALSE;
    return retVal;
}
```

C.3.4.6. \_plat\_ClockAdjustRate()

Adjust the clock rate

```c
LIB_EXPORT void
plat_ClockAdjustRate(
    int         adjust // IN: the adjust number. It could be positive
             // or negative
)
{
    // We expect the caller should only use a fixed set of constant values to
    // adjust the rate
    switch(adjust)
    {
    case CLOCK_ADJUST_COARSE:
        s_adjustRate += CLOCK_ADJUST_COARSE;
        break;
    case -CLOCK_ADJUST_COARSE:
        s_adjustRate -= CLOCK_ADJUST_COARSE;
        break;
    case CLOCK_ADJUST_MEDIUM:
        s_adjustRate += CLOCK_ADJUST_MEDIUM;
        break;
    case -CLOCK_ADJUST_MEDIUM:
        s_adjustRate -= CLOCK_ADJUST_MEDIUM;
        break;
    }
```
case CLOCK_ADJUST_FINE:
    s_adjustRate += CLOCK_ADJUST_FINE;
    break;

case -CLOCK_ADJUST_FINE:
    s_adjustRate -= CLOCK_ADJUST_FINE;
    break;

default:
    // ignore any other values;
    break;
}

if(s_adjustRate > (CLOCK_NOMINAL + CLOCK_ADJUST_LIMIT))
    s_adjustRate = CLOCK_NOMINAL + CLOCK_ADJUST_LIMIT;
if(s_adjustRate < (CLOCK_NOMINAL - CLOCK_ADJUST_LIMIT))
    s_adjustRate = CLOCK_NOMINAL - CLOCK_ADJUST_LIMIT;

return;
C.4  Entropy.c

C.4.1  Includes and Local Values

```c
#define _CRT_RAND_S
#include <stdlib.h>
#include <memory.h>
#include "PlatformData.h"
#include "Platform_fp.h"
#include <time.h>
#ifdef _MSC_VER
#include <process.h>
#else
#include <unistd.h>
#endif

extern uint32_t lastEntropy;
```

This is the last 32-bits of hardware entropy produced. We have to check to see that two consecutive 32-bit values are not the same because (according to FIPS 140-2, annex C

"If each call to a RNG produces blocks of n bits (where n > 15), the first n-bit block generated after power-up, initialization, or reset shall not be used, but shall be saved for comparison with the next n-bit block to be generated. Each subsequent generation of an n-bit block shall be compared with the previously generated block. The test shall fail if any two compared n-bit blocks are equal."

C.4.2  Functions

C.4.2.1  rand32()

Local function to get a 32-bit random number

```c
static uint32_t rand32(
    void
)
{
    uint32_t    rndNum = rand();
    #if RAND_MAX < UINT16_MAX
    // If the maximum value of the random number is a 15-bit number, then shift it up
    // 15 bits, get 15 more bits, shift that up 2 and then XOR in another value to get
    // a full 32 bits.
    rndNum = (rndNum << 15) ^ rand();
    rndNum = (rndNum << 2) ^ rand();
    #elif RAND_MAX == UINT16_MAX
    // If the maximum size is 16-bits, shift it and add another 16 bits
    rndNum = (rndNum << 16) ^ rand();
    #elif RAND_MAX < UINT32_MAX
    // If 31 bits, then shift 1 and include another random value to get the extra bit
    rndNum = (rndNum << 16) ^ rand();
    #endif
    return rndNum;
}
```
C.4.2.2. _plat__GetEntropy()

This function is used to get available hardware entropy. In a hardware implementation of this function, there would be no call to the system to get entropy.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>hardware failure of the entropy generator, this is sticky</td>
</tr>
<tr>
<td>&gt;= 0</td>
<td>the returned amount of entropy (bytes)</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT int32_t _plat__GetEntropy(
    unsigned char *entropy,   // output buffer
    uint32_t amount           // amount requested
)
{
    uint32_t rndNum;
    int32_t ret;
    // if(amount == 0)
    
    // Seed the platform entropy source if the entropy source is software. There
    // is no reason to put a guard macro (#if or #ifdef) around this code because
    // this code would not be here if someone was changing it for a system with
    // actual hardware.
    //
    // NOTE 1: The following command does not provide proper cryptographic
    // entropy. Its primary purpose to make sure that different instances of the
    // simulator, possibly started by a script on the same machine, are seeded
    // differently. Vendors of the actual TPMs need to ensure availability of
    // proper entropy using their platform-specific means.
    //
    // NOTE 2: In debug builds by default the reference implementation will seed
    // its RNG deterministically (without using any platform provided randomness).
    // See the USE_DEBUG_RNG macro and DRBG_GetEntropy() function.
    #ifdef _MSC_VER
    srand((unsigned)_plat__RealTime() ^ _getpid());
    #else
    srand((unsigned)_plat__RealTime() ^ getpid());
    #endif
    lastEntropy = rand32();
    ret = 0;
    }
    else
    {
        rndNum = rand32();
        if(rndNum == lastEntropy)
        {
            ret = -1;
        }
        else
        {
            lastEntropy = rndNum;
            // Each process will have its random number generator initialized
            // according to the process id and the initialization time. This is not a
            // lot of entropy so, to add a bit more, XOR the current time value into
            // the returned entropy value
            // NOTE: the reason for including the time here rather than have it in
            // in the value assigned to lastEntropy is that rand() could be broken and
            // using the time would in the lastEntropy value would hide this.
            rndNum ^= (uint32_t)_plat__RealTime();

            // Only provide entropy 32 bits at a time to test the ability
            // of the caller to deal with partial results.
```
88     ret = MIN(amount, sizeof(rndNum));
89     memcpy(entropy, &rndNum, ret);
90   }
91 }
92
93 return ret;
C.5  LocalityPlat.c

C.5.1. Includes

1  #include "PlatformData.h"
2  #include "Platform_fp.h"

C.5.2. Functions

C.5.2.1. _plat__LocalityGet() 

Get the most recent command locality in locality value form. This is an integer value for locality and not a locality structure. The locality can be 0-4 or 32-255. 5-31 is not allowed.

3  LIB_EXPORT unsigned char
4  _plat__LocalityGet(
5     void
6    )
7  {
8      return s_locality;
9  }

C.5.2.2. _plat__LocalitySet() 

Set the most recent command locality in locality value form

10 LIB_EXPORT void
11 _plat__LocalitySet(
12     unsigned char  locality
13    )
14  {
15      if(locality > 4 && locality < 32)
16         locality = 0;
17      s_locality = locality;
18      return;
19  }
C.6 NVMem.c

C.6.1. Description

This file contains the NV read and write access methods. This implementation uses RAM/file and does not manage the RAM/file as NV blocks. The implementation may become more sophisticated over time.

C.6.2. Includes and Local

```c
#include <memory.h>
#include <string.h>
#include <assert.h>
#include "PlatformData.h"
#include "Platform_fp.h"
#if FILE_BACKED_NV
#include <stdio.h>
FILE *s_NvFile = NULL;
#endif
```

C.6.3. Functions

C.6.3.1. NvFileOpen()

This function opens the file used to hold the NV image.

```c
static int NvFileOpen( const char *mode )
{
    // Try to open an exist NVChip file for read/write
    #if defined _MSC_VER && 1
    if(fopen_s(&s_NvFile, "NVChip", mode) != 0)
        s_NvFile = NULL;
    #else
    s_NvFile = fopen("NVChip", mode);
    #endif
    return (s_NvFile == NULL) ? -1 : 0;
}
```

C.6.3.2. NvFileCommit()

Write all of the contents of the NV image to a file.
<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>success</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>failure</td>
</tr>
</tbody>
</table>

```c
static int NvFileCommit(
    void
)
{
    int    OK;
    // If NV file is not available, return failure
    if (s_NvFile == NULL)
        return 1;
    // Write RAM data to NV
    fseek(s_NvFile, 0, SEEK_SET);
    OK = (NV_MEMORY_SIZE == fwrite(s_NV, 1, NV_MEMORY_SIZE, s_NvFile));
    OK = OK && (0 == fflush(s_NvFile));
    assert(OK);
    return OK;
}
```

C.6.3.3. NvFileSize()

This function gets the size of the NV file and puts the file pointer where desired using the seek method values. SEEK_SET => beginning; SEEK_CUR() => current position and SEEK_END => to the end of the file.

```c
static long NvFileSize(
    int    leaveAt
)
{
    long   fileSize;
    long   filePos = ftell(s_NvFile);
    //
    assert(NULL != s_NvFile);
    fseek(s_NvFile, 0, SEEK_END);
    fileSize = ftell(s_NvFile);
    switch(leaveAt)
    {
        case SEEK_SET:
            filePos = 0;
            break;
        case SEEK_CUR:
            fseek(s_NvFile, filePos, SEEK_SET);
            break;
        case SEEK_END:
            break;
        default:
            assert(FALSE);
            break;
    }
    return fileSize;
}
```

C.6.3.4. _plat__NvErrors()

This function is used by the simulator to set the error flags in the NV subsystem to simulate an error in the NV loading process.
LIB_EXPORT void _plat__NvErrors(
    int recoverable,
    int unrecoverable
)
{
    s_NV_unrecoverable = unrecoverable;
    s_NV_recoverable = recoverable;
}

C.6.3.5. _plat__NVEnable()

Enable NV memory.

This version just pulls in data from a file. In a real TPM, with NV on chip, this function would verify the integrity of the saved context. If the NV memory was not on chip but was in something like RPMB, the NV state would be read in, decrypted and integrity checked.

The recovery from an integrity failure depends on where the error occurred. It it was in the state that is discarded by TPM Reset, then the error is recoverable if the TPM is reset. Otherwise, the TPM must go into failure mode.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if success</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>if receive recoverable error</td>
</tr>
<tr>
<td>&lt;0</td>
<td>if unrecoverable error</td>
</tr>
</tbody>
</table>

LIB_EXPORT int _plat__NVEnable(
    void *platParameter // IN: platform specific parameters
)
{
    NOT_REFERENCED(platParameter); // to keep compiler quiet
    // Start assuming everything is OK
    s_NV_unrecoverable = FALSE;
    s_NV_recoverable = FALSE;
    #if FILE_BBACKED_NV
    if(s_NvFile != NULL)
        return 0;
    // Initialize all the bytes in the ram copy of the NV
    _plat__NvMemoryClear(0, NV_MEMORY_SIZE);
    // If the file exists
    if(NvFileOpen("r+b") >= 0)
    {
        long fileSize = NvFileSize(SEEK_SET); // get the file size and leave the
        // file pointer at the start
        // If the size is right, read the data
        if(NV_MEMORY_SIZE == fileSize)
            fread(s_NV, 1, NV_MEMORY_SIZE, s_NvFile);
        else
            NvFileCommit(); // for any other size, initialize it
    }
    // If NVChip file does not exist, try to create it for read/write.
    else if(NvFileOpen("w+b") >= 0)
        NvFileCommit(); // Initialize the file
    assert(NULL != s_NvFile); // Just in case we are broken for some reason.
    #endif
    // NV contents have been initialized and the error checks have been performed. For
// simulation purposes, use the signaling interface to indicate if an error is
// to be simulated and the type of the error.
if (s_NV_unrecoverable)
    return -1;
return s_NV_recoverable;

C.6.3.6. _plat__NVDisable()

Disable NV memory

LIB_EXPORT void _plat__NVDisable()
{
    void
    {
    #if FILE_BACKED_NV
        if (NULL != s_NvFile)
            fclose(s_NvFile); // Close NV file
        s_NvFile = NULL; // Set file handle to NULL
    #endif
        return;
    }

C.6.3.7. _plat__IsNvAvailable()

Check if NV is available

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NV is available</td>
</tr>
<tr>
<td>1</td>
<td>NV is not available due to write failure</td>
</tr>
<tr>
<td>2</td>
<td>NV is not available due to rate limit</td>
</tr>
</tbody>
</table>

LIB_EXPORT int _plat__IsNvAvailable()
{
    void
    {
        int     retVal = 0;
        // NV is not available if the TPM is in failure mode
        if (!s_NVIsAvailable)
            retVal = 1;
    #if FILE_BACKED_NV
        else
            retVal = (s_NvFile == NULL);
    #endif
        return retVal;
    }

C.6.3.8. _plat__NvMemoryRead()

Function: Read a chunk of NV memory

LIB_EXPORT void _plat__NvMemoryRead(
    unsigned int startOffset, // IN: read start
    unsigned int size,       // IN: size of bytes to read
    void *data                // OUT: data buffer
    )
assert(startOffset + size <= NV_MEMORY_SIZE);
memcpy(data, &s_NV[startOffset], size);    // Copy data from RAM
return;
}

C.6.3.9. _plat__NvIsDifferent()

This function checks to see if the NV is different from the test value. This is so that NV will not be written if it has not changed.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>the NV location is different from the test value</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>the NV location is the same as the test value</td>
</tr>
</tbody>
</table>

LIB_EXPORT int _plat__NvIsDifferent(
  unsigned int startOffset,       // IN: read start
  unsigned int size,              // IN: size of bytes to read
  void    *data                   // IN: data buffer
)
{
  return (memcmp(&s_NV[startOffset], data, size) != 0);
}

C.6.3.10. _plat__NvMemoryWrite()

This function is used to update NV memory. The write is to a memory copy of NV. At the end of the current command, any changes are written to the actual NV memory.

NOTE: A useful optimization would be for this code to compare the current contents of NV with the local copy and note the blocks that have changed. Then only write those blocks when _plat__NvCommit() is called.

LIB_EXPORT void _plat__NvMemoryWrite(
  unsigned int startOffset,       // IN: write start
  unsigned int size,              // IN: size of bytes to write
  void    *data                   // OUT: data buffer
)
{
  assert(startOffset + size <= NV_MEMORY_SIZE);
  memcpy(&s_NV[startOffset], data, size);    // Copy the data to the NV image
}

C.6.3.11. _plat__NvMemoryClear()

Function is used to set a range of NV memory bytes to an implementation-dependent value. The value represents the erase state of the memory.

LIB_EXPORT void _plat__NvMemoryClear(
  unsigned int start,             // IN: clear start
  unsigned int size,              // IN: number of bytes to clear
)
{
  assert(start + size <= NV_MEMORY_SIZE);
  memset(&s_NV[start], 0xff, size);  // In this implementation, assume that the erase value for NV is all 1s
}
C.6.3.12. _plat__NvMemoryMove()

Function: Move a chunk of NV memory from source to destination This function should ensure that if there overlap, the original data is copied before it is written

```c
LIB_EXPORT void
_plat__NvMemoryMove(
    unsigned int sourceOffset, // IN: source offset
    unsigned int destOffset,  // IN: destination offset
    unsigned int size         // IN: size of data being moved
)
{
    assert(sourceOffset + size <= NV_MEMORY_SIZE);
    assert(destOffset + size <= NV_MEMORY_SIZE);
    memmove(&s_NV[destOffset], &s_NV[sourceOffset], size); // Move data in RAM
    return;
}
```

C.6.3.13. _plat__NvCommit()

This function writes the local copy of NV to NV for permanent store. It will write NV_MEMORY_SIZE bytes to NV. If a file is use, the entire file is written.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NV write success</td>
</tr>
<tr>
<td>non-0</td>
<td>NV write fail</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT int
_plat__NvCommit(
    void
)
{
    #if FILE_BACKED_NV
        return (NvFileCommit() ? 0 : 1);
    #else
        return 0;
    #endif
}
```

C.6.3.14. _plat__SetNvAvail()

Set the current NV state to available. This function is for testing purpose only. It is not part of the platform NV logic

```c
LIB_EXPORT void
_plat__SetNvAvail(
    void
)
{
    s_NvIsAvailable = TRUE;
    return;
}
```

C.6.3.15. _plat__ClearNvAvail()

Set the current NV state to unavailable. This function is for testing purpose only. It is not part of the platform NV logic
LIB_EXPORT void __plat_ClearNvAvail(
    void
)
{
    s_NvIsAvailable = FALSE;
    return;
}
C.7 PowerPlat.c

C.7.1. Includes and Function Prototypes

```c
#include "PlatformData.h"
#include "Platform_fp.h"
#include "_TPM_Init_fp.h"
```

C.7.2. Functions

C.7.2.1. _plat__Signal_PowerOn()

Signal platform power on

```c
LIB_EXPORT int
_plat__Signal_PowerOn(
    void
) {
    // Reset the timer
    _plat__TimerReset();

    // Need to indicate that we lost power
    s_powerLost = TRUE;
    return 0;
}
```

C.7.2.2. _plat__WasPowerLost()

Test whether power was lost before a _TPM_Init().

This function will clear the hardware indication of power loss before return. This means that there can only be one spot in the TPM code where this value gets read. This method is used here as it is the most difficult to manage in the TPM code and, if the hardware actually works this way, it is hard to make it look like anything else. So, the burden is placed on the TPM code rather than the platform code.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>power was lost</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>power was not lost</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT int
_plat__WasPowerLost(
    void
) {
    BOOL        retVal = s_powerLost;
    s_powerLost = FALSE;
    return retVal;
}
```

C.7.2.3. _plat_Signal_Reset()

This a TPM reset without a power loss.

```c
LIB_EXPORT int
```
27 _plat__Signal_Reset(
28     void
29     )
30 {
31     // Initialize locality
32     s_locality = 0;
33
34     // Command cancel
35     s_isCanceled = FALSE;
36
37     TPM_Init();
38
39     // if we are doing reset but did not have a power failure, then we should
40     // not need to reload NV ...
41     return 0;
42 }

C.7.2.4. _plat__Signal_PowerOff()

Signal platform power off

44 LIB_EXPORT void
45 _plat__Signal_PowerOff(
46     void
47     )
48 {
49     // Prepare NV memory for power off
50     _plat__NVDisable();
51     return;
52 }

C.8 Platform_fp.h

```c
#ifndef _PLATFORM_FP_H_
#define _PLATFORM_FP_H_

C.8.1. From Cancel.c

C.8.1.1. _plat__IsCanceled()

Check if the cancel flag is set

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>if cancel flag is set</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>if cancel flag is not set</td>
</tr>
</tbody>
</table>

LIB_EXPORT int
_plat__IsCanceled(
    void
);

Set cancel flag.

LIB_EXPORT void
_plat__SetCancel(
    void
);

C.8.1.2. _plat__ClearCancel()

Clear cancel flag

LIB_EXPORT void
_plat__ClearCancel(
    void
);

C.8.2. From Clock.c

C.8.2.1. _plat__TimerReset()

This function sets current system clock time as t0 for counting TPM time. This function is called at a power on event to reset the clock. When the clock is reset, the indication that the clock was stopped is also set.

LIB_EXPORT void
_plat__TimerReset(
    void
);

C.8.2.2. _plat__TimerRestart()

This function should be called in order to simulate the restart of the timer should it be stopped while power is still applied.
C.8.2.3. _plat__RealTime()

This is another, probably futile, attempt to define a portable function that will return a 64-bit clock value that has mSec resolution.

```c
uint64_t
_plat__RealTime(
    void
);
```

C.8.2.4. _plat__TimerRead()

This function provides access to the tick timer of the platform. The TPM code uses this value to drive the TPM Clock.

The tick timer is supposed to run when power is applied to the device. This timer should not be reset by time events including _TPM_Init(). It should only be reset when TPM power is re-applied.

If the TPM is run in a protected environment, that environment may provide the tick time to the TPM as long as the time provided by the environment is not allowed to go backwards. If the time provided by the system can go backwards during a power discontinuity, then the _plat__Signal_PowerOn() should call _plat__TimerReset().

```c
LIB_EXPORT uint64_t
_plat__TimerRead(
    void
);
```

C.8.2.5. _plat__TimerWasReset()

This function is used to interrogate the flag indicating if the tick timer has been reset.

If the resetFlag parameter is SET, then the flag will be CLEAR before the function returns.

```c
LIB_EXPORT BOOL
_plat__TimerWasReset(
    void
);
```

C.8.2.6. _plat__TimerWasStopped()

This function is used to interrogate the flag indicating if the tick timer has been stopped. If so, this is typically a reason to roll the nonce.

This function will CLEAR the s_timerStopped flag before returning. This provides functionality that is similar to status register that is cleared when read. This is the model used here because it is the one that has the most impact on the TPM code as the flag can only be accessed by one entity in the TPM. Any other implementation of the hardware can be made to look like a read-once register.

```c
LIB_EXPORT BOOL
_plat__TimerWasStopped(
    void
);
```
C.8.2.7. _plat__ClockAdjustRate()

Adjust the clock rate

```c
LIB_EXPORT void _plat__ClockAdjustRate( int adjust) // IN: the adjust number. It could be positive or negative );
```

C.8.3. From Entropy.c

C.8.3.1. _plat__GetEntropy()

This function is used to get available hardware entropy. In a hardware implementation of this function, there would be no call to the system to get entropy.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0</td>
<td>hardware failure of the entropy generator, this is sticky</td>
</tr>
<tr>
<td>&gt;= 0</td>
<td>the returned amount of entropy (bytes)</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT int32_t _plat__GetEntropy( unsigned char *entropy, // output buffer uint32_t amount // amount requested );
```

C.8.4. From LocalityPlat.c

C.8.4.1. _plat__LocalityGet()

Get the most recent command locality in locality value form. This is an integer value for locality and not a locality structure. The locality can be 0-4 or 32-255. 5-31 is not allowed.

```c
LIB_EXPORT unsigned char _plat__LocalityGet( void );
```

C.8.4.2. _plat__LocalitySet()

Set the most recent command locality in locality value form

```c
LIB_EXPORT void _plat__LocalitySet( unsigned char locality );
```

C.8.5. From NVMem.c

C.8.5.1. _plat__NvErrors()

This function is used by the simulator to set the error flags in the NV subsystem to simulate an error in the NV loading process.
LIB_EXPORT void _plat__NvErrors(int recoverable, int unrecoverable);

C.8.5.2. _plat__NVEnable()

Enable NV memory.

This version just pulls in data from a file. In a real TPM, with NV on chip, this function would verify the integrity of the saved context. If the NV memory was not on chip but was in something like RPMB, the NV state would be read in, decrypted and integrity checked.

The recovery from an integrity failure depends on where the error occurred. If it was in the state that is discarded by TPM Reset, then the error is recoverable if the TPM is reset. Otherwise, the TPM must go into failure mode.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>if success</td>
</tr>
<tr>
<td>&gt; 0</td>
<td>if receive recoverable error</td>
</tr>
<tr>
<td>&lt;0</td>
<td>if unrecoverable error</td>
</tr>
</tbody>
</table>

LIB_EXPORT int _plat__NVEnable(void *platParameter // IN: platform specific parameters);

C.8.5.3. _plat__NVDisable()

Disable NV memory

LIB_EXPORT void _plat__NVDisable(void);

C.8.5.4. _plat__IsNvAvailable()

Check if NV is available

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NV is available</td>
</tr>
<tr>
<td>1</td>
<td>NV is not available due to write failure</td>
</tr>
<tr>
<td>2</td>
<td>NV is not available due to rate limit</td>
</tr>
</tbody>
</table>

LIB_EXPORT int _plat__IsNvAvailable(void);

C.8.5.5. _plat__NvMemoryRead()

Function: Read a chunk of NV memory
74 LIBEXPORT void 
75 _plat__NvMemoryRead(
76       unsigned int startOffset,  // IN: read start
77       unsigned int size,      // IN: size of bytes to read
78       void *data             // OUT: data buffer
79     );

C.8.5.6.  _plat__NvIsDifferent()

This function checks to see if the NV is different from the test value. This is so that NV will not be written if
it has not changed.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>the NV location is different from the test value</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>the NV location is the same as the test value</td>
</tr>
</tbody>
</table>

80 LIBEXPORT int 
81 _plat__NvIsDifferent(
82       unsigned int startOffset,  // IN: read start
83       unsigned int size,      // IN: size of bytes to read
84       void *data             // IN: data buffer
85     );

C.8.5.7.  _plat__NvMemoryWrite()

This function is used to update NV memory. The write is to a memory copy of NV. At the end of the
current command, any changes are written to the actual NV memory.

NOTE: A useful optimization would be for this code to compare the current contents of NV with the local copy and note
the blocks that have changed. Then only write those blocks when _plat__NvCommit() is called.

86 LIBEXPORT void 
87 _plat__NvMemoryWrite(
88       unsigned int startOffset,  // IN: write start
89       unsigned int size,      // IN: size of bytes to write
90       void *data             // OUT: data buffer
91     );

C.8.5.8.  _plat__NvMemoryClear()

Function is used to set a range of NV memory bytes to an implementation-dependent value. The value
represents the erase state of the memory.

92 LIBEXPORT void 
93 _plat__NvMemoryClear(
94       unsigned int start,      // IN: clear start
95       unsigned int size       // IN: number of bytes to clear
96     );

C.8.5.9.  _plat__NvMemoryMove()

Function: Move a chunk of NV memory from source to destination This function should ensure that if
there overlap, the original data is copied before it is written

97 LIBEXPORT void 
98 _plat__NvMemoryMove(
99       unsigned int sourceOffset,  // IN: source offset
C.8.5.10. _plat__NvCommit()

This function writes the local copy of NV to NV for permanent store. It will write NV_MEMORY_SIZE bytes to NV. If a file is use, the entire file is written.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NV write success</td>
</tr>
<tr>
<td>non-0</td>
<td>NV write fail</td>
</tr>
</tbody>
</table>

LIB_EXPORT int _plat__NvCommit(
    void
) ;

C.8.5.11. _plat__SetNvAvail()

Set the current NV state to available. This function is for testing purpose only. It is not part of the platform NV logic.

LIB_EXPORT void _plat__SetNvAvail(
    void
) ;

C.8.5.12. _plat__ClearNvAvail()

Set the current NV state to unavailable. This function is for testing purpose only. It is not part of the platform NV logic.

LIB_EXPORT void _plat__ClearNvAvail(
    void
) ;

C.8.6. From PowerPlat.c

C.8.6.1. _plat__Signal_PowerOn()

Signal platform power on

LIB_EXPORT int _plat__Signal_PowerOn(
    void
) ;

C.8.6.2. _plat__WasPowerLost()

Test whether power was lost before a _TPM_Init().

This function will clear the hardware indication of power loss before return. This means that there can only be one spot in the TPM code where this value gets read. This method is used here as it is the most
difficult to manage in the TPM code and, if the hardware actually works this way, it is hard to make it look like anything else. So, the burden is placed on the TPM code rather than the platform code.

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>power was lost</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>power was not lost</td>
</tr>
</tbody>
</table>

C.8.6.3. _plat_Signal_Reset()

This a TPM reset without a power loss.

C.8.6.4. _plat__Signal_PowerOff()

Signal platform power off

C.8.7. From PPPlat.c

C.8.7.1. _plat__PhysicalPresenceAsserted()

Check if physical presence is signaled

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>if physical presence is signaled</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>if physical presence is not signaled</td>
</tr>
</tbody>
</table>

C.8.7.2. _plat__Signal_PhysicalPresenceOn()

Signal physical presence on
C.8.7.3. \_plat\_Signal\_PhysicalPresenceOff()

Signal physical presence off

```
LIB_EXPORT void
__plat__Signal_PhysicalPresenceOff(
    void
);
```

C.8.8. From RunCommand.c

C.8.8.1. \_plat\_RunCommand()

This version of RunCommand() will set up a jum_buf and call ExecuteCommand(). If the command executes without failing, it will return and RunCommand() will return. If there is a failure in the command, then \_plat\_Fail() is called and it will longjump back to RunCommand() which will call ExecuteCommand() again. However, this time, the TPM will be in failure mode so ExecuteCommand() will simply build a failure response and return.

```
LIB_EXPORT void
__plat__RunCommand(
    uint32_t         requestSize, // IN: command buffer size
    unsigned char *request,    // IN: command buffer
    uint32_t        *responseSize, // IN/OUT: response buffer size
    unsigned char **response // IN/OUT: response buffer
);
```

C.8.8.2. \_plat\_Fail()

This is the platform depended failure exit for the TPM.

```
LIB_EXPORT NORETURN void
__plat__Fail(
    void
);
```

C.8.9. From Unique.c

C.8.10. \_plat\_GetUnique()

This function is used to access the platform-specific unique value. This function places the unique value in the provided buffer (b) and returns the number of bytes transferred. The function will not copy more data than bSize.

NOTE: If a platform unique value has unequal distribution of uniqueness and bSize is smaller than the size of the unique value, the bSize portion with the most uniqueness should be returned.

```
LIB_EXPORT uint32_t
__plat__GetUnique(
    uint32_t             which, // authorities (0) or details
    uint32_t             bSize, // size of the buffer
    unsigned char *b     // output buffer
);
```
C.9 PlatformData.h

This file contains the instance data for the Platform module. It is collected in this file so that the state of
the module is easier to manage.

```c
#ifndef _PLATFORM_DATA_H_
define _PLATFORM_DATA_H_
#include "Implementation.h"

extern int s_isCanceled;
#endif

#ifdef _MSC_VER
#include <sys/types.h>
#include <sys/timeb.h>
#else
#include <sys/time.h>
#include <time.h>
#endif

#ifndef HARDWARE_CLOCK
typedef uint64_t    clock64_t;

This is the value returned the last time that the system clock was read. This is only relevant for a
simulator or virtual TPM.

extern clock64_t     s_realTimePrevious;

These values are used to try to synthesize a long lived version of clock().

extern clock64_t     s_lastSystemTime;
extern clock64_t     s_lastReportedTime;

This is the rate adjusted value that is the equivalent of what would be read from a hardware register that
produced rate adjusted time.

extern clock64_t     s_tpmTime;
#endif // HARDWARE_CLOCK

This value indicates that the timer was reset

extern BOOL          s_timerReset;

This value indicates that the timer was stopped. It causes a clock discontinuity.

extern BOOL          s_timerStopped;

CLOCK_NOMINAL is the number of hardware ticks per mS. A value of 300000 means that the nominal
clock rate used to drive the hardware clock is 30 MHz(). The adjustment rates are used to determine the
conversion of the hardware ticks to internal hardware clock value. In practice, we would expect that there
would be a hardware register will accumulated mS. It would be incremented by the output of a pre-scaler.
The pre-scaler would divide the ticks from the clock by some value that would compensate for the
difference between clock time and real time. The code in Clock does the emulation of this function.

#define CLOCK_NOMINAL           30000

A 1% change in rate is 300 counts

#define CLOCK_ADJUST_COARSE     300
```
A 0.1% change in rate is 30 counts

```c
#define CLOCK_ADJUST_MEDIUM 30
```

A minimum change in rate is 1 count

```c
#define CLOCK_ADJUST_FINE 1
```

The clock tolerance is +/-15% (4500 counts) Allow some guard band (16.7%)

```c
#define CLOCK_ADJUST_LIMIT 5000
```

This variable records the time when plat__TimerReset() is called. This mechanism allow us to subtract the time when TPM is power off from the total time reported by clock() function

```c
extern uint64_t s_initClock;
```

This variable records the timer adjustment factor.

```c
extern unsigned int s_adjustRate;
```

For LocalityPlat.c Locality of current command

```c
extern unsigned char s_locality;
```

For NVMem.c Choose if the NV memory should be backed by RAM or by file. If this macro is defined, then a file is used as NV. If it is not defined, then RAM is used to back NV memory. Comment out to use RAM.

```c
#if (!defined VTPM) || ((VTPM != NO) && (VTPM != YES))
  # undef VTPM
  # define VTPM YES  // Default: Either YES or NO
#endif
```

For a simulation, use a file to back up the NV

```c
#if (!defined FILE_BACKED_NV) || ((FILE_BACKED_NV != NO) && (FILE_BACKED_NV != YES))
  # undef FILE_BACKED_NV
  # define FILE_BACKED_NV (VTPM && YES)  // Default: Either YES or NO
#endif
```

```c
#if SIMULATION
  # undef FILE_BACKED_NV
  # define FILE_BACKED_NV YES
#endif  // SIMULATION
```

```c
extern unsigned char s_NV[NV_MEMORY_SIZE];
extern BOOL s_NvIsAvailable;
extern BOOL s_NV_unrecoverable;
extern BOOL s_NV_recoverable;
```

For PPPlat.c Physical presence. It is initialized to FALSE

```c
extern BOOL s_physicalPresence;
```

From Power

```c
extern BOOL s_powerLost;
```

For Entropy.c

```c
extern uint32_t lastEntropy;
```
#endif // _PLATFORM_DATA_H_
C.10 PlatformData.c

C.10.1. Description

This file will instance the TPM variables that are not stack allocated. The descriptions for these variables are in Global.h for this project.

C.10.2. Includes

```c
#include "Implementation.h"
#include "PlatformData.h"
```

From Cancel.c

```c
BOOL s_isCanceled;
```

From Clock.c

```c
#include "PlatformData.h"

3 unsigned int s_adjustRate;
4 BOOL s_timerReset;
5 BOOL s_timerStopped;
6 #ifndef HARDWARE_CLOCK
7 clock64_t s_realTimePrevious;
8 clock64_t s_tpmTime;
9 clock64_t s_lastSystemTime;
10 clock64_t s_lastReportedTime;
11 #endif
```

From LocalityPlat.c

```c
unsigned char s_locality;
```

From Power.c

```c
BOOL s_powerLost;
```

From Entropy.c This values is used to determine if the entropy generator is broken. If two consecutive values are the same, then the entropy generator is considered to be broken.

```c
uint32_t lastEntropy;
```

For NVMem.c

```c
unsigned char s_NV[NV_MEMORY_SIZE];
Boolean s_NvIsAvailable;
BOOL s_NV_unrecoverable;
BOOL s_NV_recoverable;
```

From PPPlat.c

```c
BOOL s_physicalPresence;
```
C.11  PPPlat.c

C.11.1. Description

This module simulates the physical presence interface pins on the TPM.

C.11.2. Includes

```c
#include "PlatformData.h"
#include "Platform_fp.h"
```

C.11.3. Functions

C.11.3.1. _plat__PhysicalPresenceAsserted()

Check if physical presence is signaled

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE(1)</td>
<td>if physical presence is signaled</td>
</tr>
<tr>
<td>FALSE(0)</td>
<td>if physical presence is not signaled</td>
</tr>
</tbody>
</table>

```c
LIB_EXPORT int _plat__PhysicalPresenceAsserted(
    void
) {
    // Do not know how to check physical presence without real hardware.
    // so always return TRUE;
    return s_physicalPresence;
}
```

C.11.3.2. _plat__Signal_PhysicalPresenceOn()

Signal physical presence on

```c
LIB_EXPORT void _plat__Signal_PhysicalPresenceOn(
    void
) {
    s_physicalPresence = TRUE;
    return;
}
```

C.11.3.3. _plat__Signal_PhysicalPresenceOff()

Signal physical presence off

```c
LIB_EXPORT void _plat__Signal_PhysicalPresenceOff(
    void
) {
    s_physicalPresence = FALSE;
    return;
}
```
C.12 RunCommand.c

C.12.1. Introduction

This module provides the platform specific entry and fail processing. The _plat__RunCommand() function
is used to call to ExecuteCommand() in the TPM code. This function does whatever processing is
necessary to set up the platform in anticipation of the call to the TPM including setup for error
processing.

The _plat__Fail() function is called when there is a failure in the TPM. The TPM code will have set the
flag to indicate that the TPM is in failure mode. This call will then recursively call ExecuteCommand() in
order to build the failure mode response. When ExecuteCommand() returns to _plat__Fail(), the platform
will do some platform specif operation to return to the environment in which the TPM is executing. For a
simulator, setjmp/longjmp is used. For an OS, a system exit to the OS would be appropriate.

C.12.2. Includes and locals

1  #include "PlatformData.h"
2  #include "Platform_fp.h"
3  #include <setjmp.h>
4  #include "ExecCommand_fp.h"
5  jmp_buf s_jumpBuffer;

C.12.3. Functions

C.12.3.1. _plat__RunCommand()

This version of RunCommand() will set up a jum_buf and call ExecuteCommand(). If the command
executes without failing, it will return and RunCommand() will return. If there is a failure in the command,
then _plat__Fail() is called and it will longjump back to RunCommand() which will call ExecuteCommand()
again. However, this time, the TPM will be in failure mode so ExecuteCommand() will simply build a
failure response and return.

6  LIB_EXPORT void
7    _plat__RunCommand(
8        uint32_t requestSize, // IN: command buffer size
9        unsigned char *request, // IN: command buffer
10       uint32_t *responseSize, // IN/OUT: response buffer size
11       unsigned char **response // IN/OUT: response buffer
12    )
13  {
14      setjmp(s_jumpBuffer);
15      ExecuteCommand(requestSize, request, responseSize, response);
16  }

C.12.3.2. _plat__Fail()

This is the platform depended failure exit for the TPM.

17  LIB_EXPORT NORETURN void
18    _plat__Fail(
19    void
20    )
21  {
22      longjmp(&s_jumpBuffer[0], 1);
23  }
C.13 Unique.c

C.13.1. Introduction

In some implementations of the TPM, the hardware can provide a secret value to the TPM. This secret value is statistically unique to the instance of the TPM. Typical uses of this value are to provide personalization to the random number generation and as a shared secret between the TPM and the manufacturer.

C.13.2. Includes

```c
#include "PlatformData.h"
#include "Platform_fp.h"
const char notReallyUnique[] = "This is not really a unique value. A real unique value should" " be generated by the platform."
```

C.13.3. plat__GetUnique()

This function is used to access the platform-specific unique value. This function places the unique value in the provided buffer (b) and returns the number of bytes transferred. The function will not copy more data than bSize.

NOTE: If a platform unique value has unequal distribution of uniqueness and bSize is smaller than the size of the unique value, the bSize portion with the most uniqueness should be returned.

```c
LIBEXPORT uint32_t plat__GetUnique(
    uint32_t             which,       // authorities (0) or details
    uint32_t             bSize,       // size of the buffer
    unsigned char       *b           // output buffer
)
{
    const char         *from = notReallyUnique;
    uint32_t            retVal = 0;

    if (which == 0) // the authorities value
    {
        for(retVal = 0;
            *from != 0 && retVal < bSize;
            retVal++)
        {
            *b++ = *from++;
        }
    }
    else
    {
        #define uSize sizeof(notReallyUnique)
        b = &b[((bSize < uSize) ? bSize : uSize) - 1];
        for(retVal = 0;
            *from != 0 && retVal < bSize;
            retVal++)
        {
            *b-- = *from++;
        }
        return retVal;
    }
}
Annex D
(informative)
Remote Procedure Interface

D.1 Introduction

These files provide an RPC interface for a TPM simulation.

The simulation uses two ports: a command port and a hardware simulation port. Only TPM commands defined in TPM 2.0 Part 3 are sent to the TPM on the command port. The hardware simulation port is used to simulate hardware events such as power on/off and locality; and indications such as _TPM_HashStart.

D.2 Simulator_fp.h

D.2.1 From TcpServer.c

D.2.1.1 PlatformServer()

This function processes incoming platform requests.

BOOL PlatformServer(SOCKET s);

D.2.1.2 PlatformSvcRoutine()

This function is called to set up the socket interfaces to listen for commands.

DWORD WINAPI PlatformSvcRoutine(LPVOID port);

D.2.1.3 PlatformSignalService()

This function starts a new thread waiting for platform signals. Platform signals are processed one at a time in the order in which they are received.

int PlatformSignalService(int PortNumber);

D.2.1.4 RegularCommandService()

This function services regular commands.

int RegularCommandService(int PortNumber);
D.2.1.5. **StartTcpServer()**

This is the main entry-point to the TCP server. The server listens on port specified. Note that there is no way to specify the network interface in this implementation.

```c
int StartTcpServer(
    int PortNumber
);
```

D.2.1.6. **ReadBytes()**

This function reads the indicated number of bytes \((NumBytes)\) into buffer from the indicated socket.

```c
BOOL ReadBytes(
    SOCKET s,
    char *buffer,
    int NumBytes
);
```

D.2.1.7. **WriteBytes()**

This function will send the indicated number of bytes \((NumBytes)\) to the indicated socket.

```c
BOOL WriteBytes(
    SOCKET s,
    char *buffer,
    int NumBytes
);
```

D.2.1.8. **WriteUINT32()**

Send 4 bytes containinghton(1)

```c
BOOL WriteUINT32(
    SOCKET s,
    uint32_t val
);
```

D.2.1.9. **ReadVarBytes()**

Get a UINT32-length-prepended binary array. Note that the 4-byte length is in network byte order (big-endian).

```c
BOOL ReadVarBytes(
    SOCKET s,
    char *buffer,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
    uint32_t val
    SOCKET s,
D.2.1.10. WriteVarBytes()

Send a UINT32-length-prepended binary array. Note that the 4-byte length is in network byte order (big-endian).

```c
BOOL WriteVarBytes(SOCKET s,
char *buffer,
int BytesToSend);
```

D.2.1.11. TpmServer()

Processing incoming TPM command requests using the protocol / interface defined above.

```c
BOOL TpmServer(SOCKET s);
```

D.2.2. From TPMCmdp.c

D.2.2.1. Signal_PowerOn()

This function processes a power-on indication. Among other things, it calls the _TPM_Init() handler.

```c
void _rpc__Signal_PowerOn(BOOL isReset);
```

D.2.2.2. Signal_Restart()

This function processes the clock restart indication. All it does is call the platform function.

```c
void _rpc__Signal_Restart(void);
```

D.2.2.3. Signal_PowerOff()

This function processes the power off indication. Its primary function is to set a flag indicating that the next power on indication should cause _TPM_Init() to be called.

```c
void _rpc__Signal_PowerOff(void);
```

D.2.2.4. _rpc__ForceFailureMode()

This function is used to debug the Failure Mode logic of the TPM. It will set a flag in the TPM code such that the next call to TPM2_SelfTest() will result in a failure, putting the TPM into Failure Mode.
D.2.2.5. _rpc__Signal_PhysicalPresenceOn()

This function is called to simulate activation of the physical presence pin.

```c
void
_rpc__Signal_PhysicalPresenceOn(
  void
);
```

D.2.2.6. _rpc__Signal_PhysicalPresenceOff()

This function is called to simulate deactivation of the physical presence pin.

```c
void
_rpc__Signal_PhysicalPresenceOff(
  void
);
```

D.2.2.7. _rpc__Signal_Hash_Start()

This function is called to simulate a _TPM_Hash_Start() event. It will call

```c
void
_rpc__Signal_Hash_Start(
  void
);
```

D.2.2.8. _rpc__Signal_Hash_Data()

This function is called to simulate a _TPM_Hash_Data() event.

```c
void
_rpc__Signal_Hash_Data(
  _IN_BUFFER       input
);
```

D.2.2.9. _rpc__Signal_HashEnd()

This function is called to simulate a _TPM_Hash_End() event.

```c
void
_rpc__Signal_HashEnd(
  void
);
```

D.2.2.10. _rpc__Send_Command()

This is the interface to the TPM code.

```c
void
_rpc__Send_Command(
  unsigned char   locality,
```
D.2.2.11. _rpc__Signal_CancelOn()

This function is used to turn on the indication to cancel a command in process. An executing command is not interrupted. The command code may periodically check this indication to see if it should abort the current command processing and returned TPM_RC_CANCELLLED.

```c
void _rpc__Signal_CancelOn()
{
    void
}
```

D.2.2.12. _rpc__Signal_CancelOff()

This function is used to turn off the indication to cancel a command in process.

```c
void _rpc__Signal_CancelOff()
{
    void
}
```

D.2.2.13. _rpc__Signal_NvOn()

In a system where the NV memory used by the TPM is not within the TPM, the NV may not always be available. This function turns on the indicator that indicates that NV is available.

```c
void _rpc__Signal_NvOn()
{
    void
}
```

D.2.2.14. _rpc__Signal_NvOff()

This function is used to set the indication that NV memory is no longer available.

```c
void _rpc__Signal_NvOff()
{
    void
}
```

D.2.2.15. _rpc__RsaKeyCacheControl()

This function is used to enable/disable the use of the RSA key cache during simulation.

```c
void _rpc__RsaKeyCacheControl(int state)
{
    int
}
```

D.2.2.16. _rpc__Shutdown()

This function is used to stop the TPM simulator.
D.2.3. From TPMcmds.c

D.2.3.1. main()

This is the main entry point for the simulator. It registers the interface and starts listening for clients.

```c
int main(int argc, char *argv[])
```

D.2.3.2. nothing()

This function does nothing. It only exists so that there are two named functions in this section to avoid a 'dangling paragraph'.

```c
void nothing()
```

```c
#endif // _SIMULATOR_FP_H_
```
D.3  TpmTcpProtocol.h

D.3.1. Introduction

TPM commands are communicated as BYTE streams on a TCP connection. The TPM command protocol is enveloped with the interface protocol described in this file. The command is indicated by a UINT32 with one of the values below. Most commands take no parameters return no TPM errors. In these cases the TPM interface protocol acknowledges that command processing is completed by returning a UINT32=0. The command TPM_SIGNAL_HASH_DATA takes a UINT32-prepended variable length BYTE array and the interface protocol acknowledges command completion with a UINT32=0. Most TPM commands are enveloped using the TPM_SEND_COMMAND interface command. The parameters are as indicated below. The interface layer also appends a UINT32=0 to the TPM response for regularity.

D.3.2. Typedefs and Defines

```c
#ifndef TCP_TPM_PROTOCOL_H
#define TCP_TPM_PROTOCOL_H

D.3.3. TPM Commands

All commands acknowledge processing by returning a UINT32 == 0 except where noted

```c
#define TPM_SIGNAL_POWER_ON         1
#define TPM_SIGNAL_POWER_OFF        2
#define TPM_SIGNAL_PHYS_PRES_ON     3
#define TPM_SIGNAL_PHYS_PRES_OFF    4
#define TPM_SIGNAL_HASH_START       5
#define TPM_SIGNAL_HASH_DATA        6
// {UINT32 BufferSize, BYTE[BufferSize] Buffer}
#define TPM_SIGNAL_HASH_END         7
#define TPM_SEND_COMMAND            8
// {BYTE Locality, UINT32 InBufferSize, BYTE[InBufferSize] InBuffer} ->
// {UINT32 OutBufferSize, BYTE[OutBufferSize] OutBuffer}
#define TPM_SIGNAL_CANCEL_ON        9
#define TPM_SIGNAL_CANCEL_OFF       10
#define TPM_SIGNAL_NV_ON            11
#define TPM_SIGNAL_NV_OFF           12
#define TPM.Signal.KEY_CACHE_ON     13
#define TPM.Signal.KEY_CACHE_OFF    14
#define TPM_SIGNAL_REMOTE_HANDSHAKE 15
#define TPM_SIGNAL_ALTERNATIVE_RESULT 16
#define TPM_SIGNAL_RESET            17
#define TPM_SIGNAL_RESTART          18
#define TPM_SESSION_END             20
#define TPM_STOP                    21
#define TPM_GET_COMMAND_RESPONSE_SIZES 25
#define TPM_TEST_FAILURE_MODE       30
```

D.3.4. Enumerations and Structures

```c
enum TpmEndPointInfo
{
    tpmPlatformAvailable = 0x01,
    tpmUsesTbs = 0x02,
    tpmInRawMode = 0x04,
    tpmSupportsPP = 0x08
};
```

// Existing RPC interface type definitions retained so that the implementation
37 // can be re-used
38 typedef struct in_buffer
39 {
40  unsigned long BufferSize;
41  unsigned char *Buffer;
42 } _IN_BUFFER;
43
typedef unsigned char *_OUTPUT_BUFFER;
45
typedef struct out_buffer
47 {
48  uint32_t BufferSize;
49  _OUTPUT_BUFFER Buffer;
50 } _OUT_BUFFER;
51
#ifndef WIN32
52 typedef unsigned long    DWORD;
54 typedef void*            LPVOID;
55 #undef WINAPI
56#endif
58 #endif
D.4  TcpServer.c

D.4.1. Description

This file contains the socket interface to a TPM simulator.

D.4.2. Includes, Locals, Defines and Function Prototypes

To access key cache control in TPM

void RsaKeyCacheControl(int state);

D.4.3. Functions

D.4.3.1. CreateSocket()

This function creates a socket listening on PortNumber.

static int CreateSocket(
    int PortNumber,
    SOCKET *listenSocket
)
{
    WSADATA wsaData;
    struct sockaddr_in MyAddress;
    int res;

    // Initialize Winsock
    res = WSAStartup(MAKEWORD(2, 2), &wsaData);
    if(res != 0)
{  
    printf("WSAStartup failed with error: %d\n", res);
    return -1;
}
// create listening socket
*listenSocket = socket(PF_INET, SOCK_STREAM, 0);
if(INVALID_SOCKET == *listenSocket)
{
    printf("Cannot create server listen socket.  Error is 0x%x\n", WSAGetLastError());
    return -1;
}
// bind the listening socket to the specified port
ZeroMemory(&MyAddress, sizeof(MyAddress));
MyAddress.sin_port = htons((short)PortNumber);
MyAddress.sin_family = AF_INET;
res = bind(*listenSocket, (struct sockaddr*)&MyAddress, sizeof(MyAddress));
if(res == SOCKET_ERROR)
{
    printf("Bind error.  Error is 0x%x\n", WSAGetLastError());
    return -1;
}
// listen/wait for server connections
res = listen(*listenSocket, 3);
if(res == SOCKET_ERROR)
{
    printf("Listen error.  Error is 0x%x\n", WSAGetLastError());
    return -1;
}
return 0;
}

D.4.3.2. PlatformServer()

This function processes incoming platform requests.

BOOL PlatformServer(SOCKET s)
{
    BOOL OK = TRUE;
    uint32_t Command;
    //
    for(;;)
    {
        OK = ReadBytes(s, (char*)&Command, 4);
        // client disconnected (or other error).  We stop processing this client
        // and return to our caller who can stop the server or listen for another
        // connection.
        if(!OK) return TRUE;
        Command = ntohl(Command);
        switch(Command)
        {
        case TPM_SIGNAL_POWER_ON:
            _rpc__Signal_PowerOn(FALSE);
            break;
        case TPM_SIGNAL_POWER_OFF:
            _rpc__Signal_PowerOff();
            break;
        case TPM SIGNAL_RESET:
            _rpc__Signal_PowerOn(TRUE);
            break;
D.4.3.3. PlatformSvcRoutine()

This function is called to set up the socket interfaces to listen for commands.

```c
DWORD WINAPI PlatformSvcRoutine(
    LPVOID port
)
{
    case TPM_SIGNAL_RESTART:
        _rpc__Signal_Restart();
        break;
    case TPM_SIGNAL_PHYS_PRES_ON:
        _rpc__Signal_PhysicalPresenceOn();
        break;
    case TPM_SIGNAL_PHYS_PRES_OFF:
        _rpc__Signal_PhysicalPresenceOff();
        break;
    case TPM_SIGNAL_CANCEL_ON:
        _rpc__Signal_CancelOn();
        break;
    case TPM_SIGNAL_CANCEL_OFF:
        _rpc__Signal_CancelOff();
        break;
    case TPM_SIGNAL_NV_ON:
        _rpc__Signal_NvOn();
        break;
    case TPM_SIGNAL_NV_OFF:
        _rpc__Signal_NvOff();
        break;
    case TPM_SIGNAL_KEY_CACHE_ON:
        _rpc__RsaKeyCacheControl(TRUE);
        break;
    case TPM_SIGNAL_KEY_CACHE_OFF:
        _rpc__RsaKeyCacheControl(FALSE);
        break;
    case TPM_SESSION_END:
        // Client signaled end-of-session
        TpmEndSimulation();
        return TRUE;
    case TPM_STOP:
        // Client requested the simulator to exit
        return FALSE;
    case TPM_TEST_FAILURE_MODE:
        _rpc__ForceFailureMode();
        break;
    case TPM_GET_COMMAND_RESPONSE_SIZES:
        OK = WriteVarBytes(s, (char *)&CommandResponseSizes,
            sizeof(CommandResponseSizes));
        memset(&CommandResponseSizes, 0, sizeof(CommandResponseSizes));
        if(!OK)
            return TRUE;
        break;
    default:
        printf("Unrecognized platform interface command %d\n",
            (int)Command);
        WriteUINT32(s, 1);
        return TRUE;
    }
    WriteUINT32(s, 0);
    return FALSE;
}
```
int PortNumber = (int)(INT_PTR)port;
SOCKET listenSocket, serverSocket;
struct sockaddr_in HerAddress;
int res;
int length;
BOOL continueServing;

// res = CreateSocket(PortNumber, &listenSocket);
if (res != 0) {
    printf("Create platform service socket fail\n");
    return res;
}

// Loop accepting connections one-by-one until we are killed or asked to stop
// Note the platform service is single-threaded so we don’t listen for a new
// connection until the prior connection drops.
do {
    printf("Platform server listening on port %d\n", PortNumber);
    // blocking accept
    length = sizeof(HerAddress);
    serverSocket = accept(listenSocket, (struct sockaddr*) &HerAddress, &length);
    if(serverSocket == SOCKET_ERROR) {
        printf("Accept error. Error is 0x%x\n", WSAGetLastError());
        return -1;
    }
    printf("Client accepted\n");
    // normal behavior on client disconnection is to wait for a new client
    // to connect
    continueServing = PlatformServer(serverSocket);
    closesocket(serverSocket);
} while(continueServing);

return 0;

D.4.3.4. PlatformSignalService()

This function starts a new thread waiting for platform signals. Platform signals are processed one at a time in the order in which they are received.

int PlatformSignalService(
    int PortNumber
) {
    HANDLE hPlatformSvc;
    int ThreadId;
    int port = PortNumber;
    // Create service thread for platform signals
    hPlatformSvc = CreateThread(NULL, 0,
        (LPTHREAD_START_ROUTINE)PlatformSvcRoutine,
        (LPVOID)(INT_PTR)port, 0, (LPDWORD)&ThreadId);
    if(hPlatformSvc == NULL) {
        printf("Thread Creation failed\n");
        return -1;
    }
}
D.4.3.5. RegularCommandService()

This function services regular commands.

```c
int
RegularCommandService(
    int PortNumber)
{
    SOCKET listenSocket;
    SOCKET serverSocket;
    struct sockaddr_in HerAddress;
    int res, length;
    BOOL continueServing;

    // res = CreateSocket(PortNumber, &listenSocket);
    if(res != 0)
    {
        printf("Create platform service socket fail\n");
        return res;
    }
    // Loop accepting connections one-by-one until we are killed or asked to stop
    // Note that the TPM command service is single-threaded so we don't listen for
    // a new connection until the prior connection drops.
    do
    {
        printf("TPM command server listening on port %d\n", PortNumber);
        // blocking accept
        length = sizeof(HerAddress);
        serverSocket = accept(listenSocket, (struct sockaddr*) &HerAddress, &length);
        if(serverSocket == SOCKET_ERROR)
        {
            printf("Accept error. Error is 0x%x\n", WSAGetLastError());
            return -1;
        }
        printf("Client accepted\n");
        // normal behavior on client disconnection is to wait for a new client
        // to connect
        continueServing = TpmServer(serverSocket);
        closesocket(serverSocket);
    } while(continueServing);
    return 0;
}
```

D.4.3.6. StartTcpServer()

This is the main entry-point to the TCP server. The server listens on port specified.

Note that there is no way to specify the network interface in this implementation.

```c
int
StartTcpServer(
    int PortNumber)
{
    int res;
```
D.4.3.7. ReadBytes()

This function reads the indicated number of bytes (NumBytes) into buffer from the indicated socket.

```c
BOOL
ReadBytes(
    SOCKET           s,
    char             *buffer,
    int              NumBytes
)
{
    int             res;
    int             numGot = 0;
    //
    while(numGot < NumBytes)
    {
        res = recv(s, buffer + numGot, NumBytes - numGot, 0);
        if(res == -1)
        {
            printf("Receive error. Error is 0x%x\n", WSAGetLastError());
            return FALSE;
        }
        if(res == 0)
        {
            return FALSE;
        }
        numGot += res;
    }
    return TRUE;
}
```

D.4.3.8. WriteBytes()

This function will send the indicated number of bytes (NumBytes) to the indicated socket.

```c
BOOL
WriteBytes(
    SOCKET           s,
    char             *buffer,
    int              NumBytes
)
{
    int             res;
    int             numSent = 0;
    //
```
D.4.3.9. WriteUINT32()

Send 4 bytes containing htonl(1)

```c
BOOL WriteUINT32(
    SOCKET s,
    uint32_t val
) {
    UINT32 netVal = htonl(val);
    return WriteBytes(s, (char*)&netVal, 4);
}
```

D.4.3.10. ReadVarBytes()

Get a UINT32-length-prepended binary array. Note that the 4-byte length is in network byte order (big-endian).

```c
BOOL ReadVarBytes(
    SOCKET s,
    char *buffer,
    uint32_t *BytesReceived,
    int MaxLen
) {
    //
    res = ReadBytes(s, (char*)&length, 4);
    if(!res) return res;
    length = ntohl(length);
    *BytesReceived = length;
    if(length > MaxLen)
        printf("Buffer too big. Client says %d\n", length);
    return FALSE;
}
```
D.4.3.11. WriteVarBytes()

Send a UINT32-length-prepended binary array. Note that the 4-byte length is in network byte order (big-endian).

```c
BOOL WriteVarBytes(
    SOCKET s,
    char *buffer,
    int BytesToSend
)
{
    uint32_t netLength = htonl(BytesToSend);
    BOOL res;
    //
    res = WriteBytes(s, (char*)&netLength, 4);
    if(!res)
        return res;
    res = WriteBytes(s, buffer, BytesToSend);
    if(!res)
        return res;
    return TRUE;
}
```

D.4.3.12. TpmServer()

Processing incoming TPM command requests using the protocol / interface defined above.

```c
BOOL TpmServer(
    SOCKET s
)
{
    uint32_t length;
    uint32_t Command;
    BYTE locality;
    BOOL OK;
    int result;
    int clientVersion;
    _IN_BUFFER InBuffer;
    _OUT_BUFFER OutBuffer;
    //
    for(;;)
    {
        OK = ReadBytes(s, (char*)&Command, 4);
        // client disconnected (or other error). We stop processing this client
        // and return to our caller who can stop the server or listen for another
        // connection.
        if(!OK)
            return TRUE;
        Command = ntohl(Command);
        switch(Command)
        {
            case TPM_SIGNAL_HASH_START:
                _rpc__Signal_Hash_Start();
                break;
            case TPM_SIGNAL_HASH_END:
                _rpc__Signal_HashEnd();
                break;
            case TPM_SIGNAL_HASH_DATA:
OK = ReadVarBytes(s, InputBuffer, &length, MAX_BUFFER);
if (!OK) return TRUE;
InBuffer.Buffer = (BYTE*)InputBuffer;
InBuffer.BufferSize = length;
_rpc__Signal_Hash_Data(InBuffer);
between;
case TPM_SEND_COMMAND:
OK = ReadBytes(s, (char*)&locality, 1);
if (!OK)
return TRUE;
OK = ReadVarBytes(s, InputBuffer, &length, MAX_BUFFER);
if (!OK)
return TRUE;
InBuffer.Buffer = (BYTE*)InputBuffer;
InBuffer.BufferSize = length;
OutBuffer.BufferSize = MAX_BUFFER;
OutBuffer.Buffer = (_OUTPUT_BUFFER)OutputBuffer;
// record the number of bytes in the command if it is the largest
// we have seen so far.
if (InBuffer.BufferSize > CommandResponseSizes.largestCommandSize)
{
    CommandResponseSizes.largestCommandSize = InBuffer.BufferSize;
    memcpy(&CommandResponseSizes.largestCommand,
           &InputBuffer[6], sizeof(UINT32));
}
_rpc__Send_Command(locality, InBuffer, &OutBuffer);
// record the number of bytes in the response if it is the largest
// we have seen so far.
if (OutBuffer.BufferSize > CommandResponseSizes.largestResponseSize)
{
    CommandResponseSizes.largestResponseSize
        = OutBuffer.BufferSize;
    memcpy(&CommandResponseSizes.largestResponse,
           &OutputBuffer[6], sizeof(UINT32));
}
OK = WriteVarBytes(s,
   (char*)OutBuffer.Buffer,
   OutBuffer.BufferSize);
if (!OK)
return TRUE;
between;
case TPM_REMOTE_HANDSHAKE:
OK = ReadBytes(s, (char*)&clientVersion, 4);
if (!OK)
return TRUE;
if (clientVersion == 0)
{
    printf("Unsupported client version (0).\n");
    return TRUE;
}
OK &= WriteUINT32(s, ServerVersion);
OK &= WriteUINT32(s, tpmInRawMode
        | tpmPlatformAvailable | tpmSupportsPP);
between;
case TPM_SET_ALTERNATIVE_RESULT:
OK = ReadBytes(s, (char*)&result, 4);
if (!OK)
return TRUE;
// Alternative result is not applicable to the simulator.
between;
case TPM_SESSION_END:
// Client signaled end-of-session
return TRUE;
between;
case TPM_STOP:
// Client requested the simulator to exit
return FALSE;
default:
    printf("Unrecognized TPM interface command %d\n", (int)Command);
    return TRUE;
}
OK = WriteUINT32(s, 0);
if(!OK)
    return TRUE;
return FALSE;
D.5 TPMcmdp.c

D.5.1. Description

This file contains the functions that process the commands received on the control port or the command port of the simulator. The control port is used to allow simulation of hardware events (such as, _TPM_Hash_Start()) to test the simulated TPM's reaction to those events. This improves code coverage of the testing.

D.5.2. Includes and Data Definitions

```c
#include <stdlib.h>
#include <stdio.h>
#include <setjmp.h>
#include "TpmBuildSwitches.h"
#include <windows.h>
#include <winsock.h>
#include "Platform_fp.h"
#include "ExecCommand_fp.h"
#include "Manufacture_fp.h"
#include "TPM_Init_fp.h"
#include "TPM_Hash_Start_fp.h"
#include "TPM_Hash_Data_fp.h"
#include "TPM_Hash_End_fp.h"
#include "TpmFail_fp.h"
#include "TpmTcpProtocol.h"
#include "Simulator_fp.h"

static BOOL s_isPowerOn = FALSE;
```

D.5.3. Functions

D.5.3.1. Signal_PowerOn()

This function processes a power-on indication. Among other things, it calls the _TPM_Init() handler.

```c
void _rpc__Signal_PowerOn(
  BOOL isReset )
{
  // if power is on and this is not a call to do TPM reset then return
  if(s_isPowerOn && !isReset)
  return;

  // If this is a reset but power is not on, then return
  if(isReset && !s_isPowerOn)
  return;

  // Unless this is just a reset, pass power on signal to platform
  if(!isReset)
  _plat__Signal_PowerOn();

  // Power on and reset both lead to _TPM_Init()
  _plat__Signal_Reset();

  // Set state as power on
  s_isPowerOn = TRUE;
}
```
D.5.3.2. Signal_Restart()

This function processes the clock restart indication. All it does is call the platform function.

```c
void _rpc__Signal_Restart(
  void
)
{
  _plat__TimerRestart();
}
```

D.5.3.3. Signal_PowerOff()

This function processes the power off indication. Its primary function is to set a flag indicating that the next power on indication should cause _TPM_Init() to be called.

```c
void _rpc__Signal_PowerOff(
  void
)
{
  if(!s_isPowerOn) return;
  // Pass power off signal to platform
  _plat__Signal_PowerOff();
  s_isPowerOn = FALSE;
  return;
}
```

D.5.3.4. _rpc__ForceFailureMode()

This function is used to debug the Failure Mode logic of the TPM. It will set a flag in the TPM code such that the next call to TPM2_SelfTest() will result in a failure, putting the TPM into Failure Mode.

```c
void _rpc__ForceFailureMode(
  void
)
{
  SetForceFailureMode();
}
```

D.5.3.5. _rpc__Signal_PhysicalPresenceOn()

This function is called to simulate activation of the physical presence pin.

```c
void _rpc__Signal_PhysicalPresenceOn(
  void
)
{
  // If TPM is power off, reject this signal
  if(!s_isPowerOn) return;
  // Pass physical presence on to platform
  _plat__Signal_PhysicalPresenceOn();
}
D.5.3.6. _rpc__Signal_PhysicalPresenceOff()

This function is called to simulate deactivation of the physical presence pin.

```c
void _rpc__Signal_PhysicalPresenceOff()
{
    // If TPM is power off, reject this signal
    if(!s_isPowerOn) return;
    // Pass physical presence off to platform
    _plat__Signal_PhysicalPresenceOff();
    return;
}
```

D.5.3.7. _rpc__Signal_Hash_Start()

This function is called to simulate a _TPM_Hash_Start() event. It will call

```c
void _rpc__Signal_Hash_Start()
{
    // If TPM is power off, reject this signal
    if(!s_isPowerOn) return;
    // Pass _TPM_Hash_Start signal to TPM
    _TPM_Hash_Start();
    return;
}
```

D.5.3.8. _rpc__Signal_Hash_Data()

This function is called to simulate a _TPM_Hash_Data() event.

```c
void _rpc__Signal_Hash_Data(_IN_BUFFER input)
{
    // If TPM is power off, reject this signal
    if(!s_isPowerOn) return;
    // Pass _TPM_Hash_Data signal to TPM
    _TPM_Hash_Data(input.BufferSize, input.Buffer);
    return;
}
```

D.5.3.9. _rpc__Signal_HashEnd()

This function is called to simulate a _TPM_Hash_End() event.

```c
void _rpc__Signal_HashEnd()
{
}
```
D.5.3.10. _rpc__Signal_HashEnd()

This is the interface to the TPM code.

void
_rpc__Signal_HashEnd()
{
    void
    
    {  // If TPM is power off, reject this signal
        if(!s_isPowerOn) return;
        // Pass _TPM_HashEnd signal to TPM
        _TPM_Hash_End();
        return;
    }
}

D.5.3.11. _rpc__Signal_CancelOn()

This function is used to turn on the indication to cancel a command in process. An executing command is not interrupted. The command code may periodically check this indication to see if it should abort the current command processing and returned TPM_RC_CANCELLED.

void
_rpc__Signal_CancelOn()
{
    void
    
    {  // If TPM is power off, reject this signal
        if(!s_isPowerOn) return;
        // Set the platform canceling flag.
        _plat__SetCancel();
        return;
    }
}

D.5.3.12. _rpc__Signal_CancelOff()

This function is used to turn off the indication to cancel a command in process.
void _rpc__Signal_CancelOff(
    void
)
{
    // If TPM is power off, reject this signal
    if(!s_isPowerOn) return;
    // Set the platform canceling flag.
    _plat__ClearCancel();
    return;
}

D.5.3.13. _rpc__Signal_NvOn()

In a system where the NV memory used by the TPM is not within the TPM, the NV may not always be available. This function turns on the indicator that indicates that NV is available.

void _rpc__Signal_NvOn(
    void
)
{
    // If TPM is power off, reject this signal
    if(!s_isPowerOn) return;
    _plat__SetNvAvail();
    return;
}

D.5.3.14. _rpc__Signal_NvOff()

This function is used to set the indication that NV memory is no longer available.

void _rpc__Signal_NvOff(
    void
)
{
    // If TPM is power off, reject this signal
    if(!s_isPowerOn) return;
    _plat__ClearNvAvail();
    return;
}

void RsaKeyCacheControl(int state);

D.5.3.15. _rpc__RsaKeyCacheControl()

This function is used to enable/disable the use of the RSA key cache during simulation.

void _rpc__RsaKeyCacheControl(
    int state
)
{
    #if USE_RSA_KEY_CACHE
    RsaKeyCacheControl(state);
    #else
    NOT_REFERENCED(state);
    #endif
D.5.3.16. _rpc__Shutdown()

This function is used to stop the TPM simulator.

```c
void _rpc__Shutdown(
    void
) {
    RPC_STATUS status;
    // Stop TPM
    TPM_TearDown();
    status = RpcMgmtStopServerListening(NULL);
    if(status != RPC_S_OK)
    {
        printf("RpcMgmtStopServerListening returned: 0x%x\n", status);
        exit(status);
    }
    status = RpcServerUnregisterIf(NULL, NULL, FALSE);
    if(status != RPC_S_OK)
    {
        printf("RpcServerUnregisterIf returned 0x%x\n", status);
        exit(status);
    }
    return;
}
```
D.6 TPMCmds.c

D.6.1. Description

This file contains the entry point for the simulator.

D.6.2. Includes, Defines, Data Definitions, and Function Prototypes

```c
#include "TpmBuildSwitches.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <ctype.h>
#include <string.h>
#include <windows.h>
#include <winsock.h>
#include "TpmTcpProtocol.h"
#include "Manufacture_fp.h"
#include "Platform_fp.h"
#include "Simulator_fp.h"
#define PURPOSE \\
"TPM Reference Simulator.\nCopyright Microsoft Corp.\n"
#define DEFAULT_TPM_PORT 2321
void* MainPointer;
```

D.6.3. Functions

D.6.3.1. Usage()

This function prints the proper calling sequence for the simulator.

```c
static void Usage(
    char *pszProgramName
) {
    fprintf(stderr, "%s", PURPOSE);
    fprintf(stderr, "Usage:
    
    %s - Starts the TPM server listening on port %d\n",
            pszProgramName, DEFAULT_TPM_PORT);
    fprintf(stderr, "%s PortNum - Starts the TPM server listening on port PortNum\n",
            pszProgramName);
    fprintf(stderr, "%s ? - This message\n", pszProgramName);
    exit(1);
}
```

D.6.3.2. main()

This is the main entry point for the simulator. It registers the interface and starts listening for clients.

```c
int main(
    int argc,
    char *argv[]
) {
    int portNum = DEFAULT_TPM_PORT;
    if(argc > 2)
```
{  Usage(argv[0]);
}

if(argc == 2)
{
  if(strcmp(argv[1], "?") == 0)
  {
    Usage(argv[0]);
  }
  portNum = atoi(argv[1]);
  if(portNum <= 0 || portNum > 65535)
  {
    Usage(argv[0]);
  }
}
 Plat_NVEnable(NULL);

if(TPM_Manufacture(1) != 0)
{
  exit(1);
}

// Coverage test - repeated manufacturing attempt
if(TPM_Manufacture(0) != 1)
{
  exit(2);
}

// Coverage test - re-manufacturing
TPM_TearDown();
if(TPM_Manufacture(1) != 0)
{
  exit(3);
}

// Disable NV memory
_plat__NVDisable();
StartTcpServer(portNum);
return EXIT_SUCCESS;

D.6.3.3. nothing()

This function does nothing. It only exists so that there are two named functions in this section to avoid a 'dangling paragraph'.

void nothing(
  void
)