Licenses and Notices

1. **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.

2. **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.

3. **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 owner.
CONTENTS

1 Scope ......................................................................................................................... 1
2 Terms and Definitions ................................................................................................. 1
3 Symbols and abbreviated terms ..................................................................................... 1
4 Notation ......................................................................................................................... 1

4.1 Introduction ................................................................................................................. 1
4.2 Table Decorations ......................................................................................................... 1
4.3 Handle and Parameter Demarcation ............................................................................. 3
4.4 AuthorizationSize and ParameterSize ........................................................................ 3

5 Normative References .................................................................................................. 4
6 Symbols and Abbreviated Terms ..................................................................................... 4
7 Command Processing ..................................................................................................... 4

7.1 Introduction ............................................................................................................... 4
7.2 Command Header Validation ....................................................................................... 4
7.3 Mode Checks .............................................................................................................. 4
7.4 Handle Area Validation ............................................................................................... 5
7.5 Session Area Validation .............................................................................................. 6
7.6 Authorization Checks ................................................................................................. 7
7.7 Parameter Decryption ................................................................................................. 8
7.8 Parameter Unmarshaling ............................................................................................. 9
7.9 Command Post Processing ......................................................................................... 10

8 Response Values ......................................................................................................... 12

8.1 Tag ........................................................................................................................... 12
8.2 Response Codes ......................................................................................................... 12

9 Implementation Dependent .......................................................................................... 15
10 Detailed Actions Assumptions ..................................................................................... 16

10.1 Introduction .............................................................................................................. 16
10.2 Pre-processing ......................................................................................................... 16
10.3 Post Processing ....................................................................................................... 16

11 Start-up ...................................................................................................................... 17

11.1 Introduction .............................................................................................................. 17
11.2 _TPM_Init ................................................................................................................. 17
11.3 TPM2_Startup .......................................................................................................... 19
11.4 TPM2_Shutdown ...................................................................................................... 24

12 Testing ........................................................................................................................ 28

12.1 Introduction .............................................................................................................. 28
12.2 TPM2_SelfTest ......................................................................................................... 29
12.3 TPM2_IncrementalSelfTest ..................................................................................... 32
12.4 TPM2_GetTestResult ............................................................................................... 35

13 Session Commands ................................................................................................... 38

13.1 TPM2_StartAuthSession ......................................................................................... 38
13.2 TPM2_PolicyRestart .............................................................................................. 43

14 Object Commands ...................................................................................................... 46
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Part 3: Commands</td>
<td></td>
</tr>
<tr>
<td>14.1 TPM2_Create</td>
<td>46</td>
</tr>
<tr>
<td>14.2 TPM2_Load</td>
<td>51</td>
</tr>
<tr>
<td>14.3 TPM2_LoadExternal</td>
<td>55</td>
</tr>
<tr>
<td>14.4 TPM2_ReadPublic</td>
<td>60</td>
</tr>
<tr>
<td>14.5 TPM2_ActivateCredential</td>
<td>63</td>
</tr>
<tr>
<td>14.6 TPM2_MakeCredential</td>
<td>67</td>
</tr>
<tr>
<td>14.7 TPM2_Unseal</td>
<td>70</td>
</tr>
<tr>
<td>14.8 TPM2_ObjectChangeAuth</td>
<td>73</td>
</tr>
<tr>
<td>15 Duplication Commands</td>
<td>77</td>
</tr>
<tr>
<td>15.1 TPM2_Duplicate</td>
<td>77</td>
</tr>
<tr>
<td>15.2 TPM2_Rewrap</td>
<td>81</td>
</tr>
<tr>
<td>15.3 TPM2_Import</td>
<td>86</td>
</tr>
<tr>
<td>16 Asymmetric Primitives</td>
<td>92</td>
</tr>
<tr>
<td>16.1 Introduction</td>
<td>92</td>
</tr>
<tr>
<td>16.2 TPM2_RSA_Encrypt</td>
<td>92</td>
</tr>
<tr>
<td>16.3 TPM2_RSA_Decrypt</td>
<td>97</td>
</tr>
<tr>
<td>16.4 TPM2_ECDH_KeyGen</td>
<td>101</td>
</tr>
<tr>
<td>16.5 TPM2_ECDH_ZGen</td>
<td>104</td>
</tr>
<tr>
<td>16.6 TPM2_ECC_Parameters</td>
<td>107</td>
</tr>
<tr>
<td>16.7 TPM2_ZGen_2Phase</td>
<td>108</td>
</tr>
<tr>
<td>17 Symmetric Primitives</td>
<td>113</td>
</tr>
<tr>
<td>17.1 Introduction</td>
<td>113</td>
</tr>
<tr>
<td>17.2 TPM2_EncryptDecrypt</td>
<td>115</td>
</tr>
<tr>
<td>17.3 TPM2_Hash</td>
<td>119</td>
</tr>
<tr>
<td>17.4 TPM2_HMAC</td>
<td>122</td>
</tr>
<tr>
<td>18 Random Number Generator</td>
<td>126</td>
</tr>
<tr>
<td>18.1 TPM2_GetRandom</td>
<td>126</td>
</tr>
<tr>
<td>18.2 TPM2_StirRandom</td>
<td>129</td>
</tr>
<tr>
<td>19 Hash/HMAC/Event Sequences</td>
<td>132</td>
</tr>
<tr>
<td>19.1 Introduction</td>
<td>132</td>
</tr>
<tr>
<td>19.2 TPM2_HMAC_Start</td>
<td>132</td>
</tr>
<tr>
<td>19.3 TPM2_HashSequenceStart</td>
<td>136</td>
</tr>
<tr>
<td>19.4 TPM2_SequenceUpdate</td>
<td>139</td>
</tr>
<tr>
<td>19.5 TPM2_SequenceComplete</td>
<td>143</td>
</tr>
<tr>
<td>19.6 TPM2_EventSequenceComplete</td>
<td>147</td>
</tr>
<tr>
<td>20 Attestation Commands</td>
<td>151</td>
</tr>
<tr>
<td>20.1 Introduction</td>
<td>151</td>
</tr>
<tr>
<td>20.2 TPM2_Certify</td>
<td>153</td>
</tr>
<tr>
<td>20.3 TPM2_CertifyCreation</td>
<td>157</td>
</tr>
<tr>
<td>20.4 TPM2_Quote</td>
<td>161</td>
</tr>
<tr>
<td>20.5 TPM2_GetSessionAuditDigest</td>
<td>165</td>
</tr>
<tr>
<td>20.6 TPM2_GetCommandAuditDigest</td>
<td>169</td>
</tr>
<tr>
<td>20.7 TPM2_GetTime</td>
<td>173</td>
</tr>
<tr>
<td>21 Ephemeral EC Keys</td>
<td>177</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>21.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>21.2</td>
<td>TPM2_Commit</td>
</tr>
<tr>
<td>21.3</td>
<td>TPM2_EC_Ephemeral</td>
</tr>
<tr>
<td>22</td>
<td>Signing and Signature Verification</td>
</tr>
<tr>
<td>22.1</td>
<td>TPM2_VerifySignature</td>
</tr>
<tr>
<td>22.2</td>
<td>TPM2_Sign</td>
</tr>
<tr>
<td>23</td>
<td>Command Audit</td>
</tr>
<tr>
<td>23.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>23.2</td>
<td>TPM2_SetCommandCodeAuditStatus</td>
</tr>
<tr>
<td>24</td>
<td>Integrity Collection (PCR)</td>
</tr>
<tr>
<td>24.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>24.2</td>
<td>TPM2_PCR_Extend</td>
</tr>
<tr>
<td>24.3</td>
<td>TPM2_PCR_Event</td>
</tr>
<tr>
<td>24.4</td>
<td>TPM2_PCR_Read</td>
</tr>
<tr>
<td>24.5</td>
<td>TPM2_PCR_Allocate</td>
</tr>
<tr>
<td>24.6</td>
<td>TPM2_PCR_SetAuthPolicy</td>
</tr>
<tr>
<td>24.7</td>
<td>TPM2_PCR_SetAuthValue</td>
</tr>
<tr>
<td>24.8</td>
<td>TPM2_PCR_Reset</td>
</tr>
<tr>
<td>24.9</td>
<td>_TPM_Hash_Start</td>
</tr>
<tr>
<td>24.10</td>
<td>_TPM_Hash_Data</td>
</tr>
<tr>
<td>24.11</td>
<td>_TPM_Hash_End</td>
</tr>
<tr>
<td>25</td>
<td>Enhanced Authorization (EA) Commands</td>
</tr>
<tr>
<td>25.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>25.2</td>
<td>Signed Authorization Actions</td>
</tr>
<tr>
<td>25.3</td>
<td>TPM2_PolicySigned</td>
</tr>
<tr>
<td>25.4</td>
<td>TPM2_PolicySecret</td>
</tr>
<tr>
<td>25.5</td>
<td>TPM2_PolicyTicket</td>
</tr>
<tr>
<td>25.6</td>
<td>TPM2_PolicyOR</td>
</tr>
<tr>
<td>25.7</td>
<td>TPM2_PolicyPCR</td>
</tr>
<tr>
<td>25.8</td>
<td>TPM2_PolicyLocality</td>
</tr>
<tr>
<td>25.9</td>
<td>TPM2_PolicyNV</td>
</tr>
<tr>
<td>25.10</td>
<td>TPM2_PolicyCounterTimer</td>
</tr>
<tr>
<td>25.11</td>
<td>TPM2_PolicyCommandCode</td>
</tr>
<tr>
<td>25.12</td>
<td>TPM2_PolicyPhysicalPresence</td>
</tr>
<tr>
<td>25.13</td>
<td>TPM2_PolicyCpHash</td>
</tr>
<tr>
<td>25.14</td>
<td>TPM2_PolicyNameHash</td>
</tr>
<tr>
<td>25.15</td>
<td>TPM2_PolicyDuplicationSelect</td>
</tr>
<tr>
<td>25.16</td>
<td>TPM2_PolicyAuthorize</td>
</tr>
<tr>
<td>25.17</td>
<td>TPM2_PolicyAuthValue</td>
</tr>
<tr>
<td>25.18</td>
<td>TPM2_PolicyPassword</td>
</tr>
<tr>
<td>25.19</td>
<td>TPM2_PolicyGetDigest</td>
</tr>
<tr>
<td>25.20</td>
<td>TPM2_PolicyNvWritten</td>
</tr>
<tr>
<td>26</td>
<td>Hierarchy Commands</td>
</tr>
<tr>
<td>26.1</td>
<td>TPM2_CreatePrimary</td>
</tr>
<tr>
<td>26.2</td>
<td>TPM2_HierarchyControl</td>
</tr>
<tr>
<td>26.3</td>
<td>TPM2_SetPrimaryPolicy</td>
</tr>
<tr>
<td>Section</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>26.4</td>
<td>TPM2_ChangePPS</td>
</tr>
<tr>
<td>26.5</td>
<td>TPM2_ChangeEPS</td>
</tr>
<tr>
<td>26.6</td>
<td>TPM2_Clear</td>
</tr>
<tr>
<td>26.7</td>
<td>TPM2_ClearControl</td>
</tr>
<tr>
<td>26.8</td>
<td>TPM2_HierarchyChangeAuth</td>
</tr>
<tr>
<td>27</td>
<td>Dictionary Attack Functions</td>
</tr>
<tr>
<td>27.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>27.2</td>
<td>TPM2_DictionaryAttackLockReset</td>
</tr>
<tr>
<td>27.3</td>
<td>TPM2_DictionaryAttackParameters</td>
</tr>
<tr>
<td>28</td>
<td>Miscellaneous Management Functions</td>
</tr>
<tr>
<td>28.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>28.2</td>
<td>TPM2_PP_Commands</td>
</tr>
<tr>
<td>28.3</td>
<td>TPM2_SetAlgorithmSet</td>
</tr>
<tr>
<td>29</td>
<td>Field Upgrade</td>
</tr>
<tr>
<td>29.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>29.2</td>
<td>TPM2_FieldUpgradeStart</td>
</tr>
<tr>
<td>29.3</td>
<td>TPM2_FieldUpgradeData</td>
</tr>
<tr>
<td>29.4</td>
<td>TPM2_FirmwareRead</td>
</tr>
<tr>
<td>30</td>
<td>Context Management</td>
</tr>
<tr>
<td>30.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>30.2</td>
<td>TPM2_ContextSave</td>
</tr>
<tr>
<td>30.3</td>
<td>TPM2_ContextLoad</td>
</tr>
<tr>
<td>30.4</td>
<td>TPM2_FlushContext</td>
</tr>
<tr>
<td>30.5</td>
<td>TPM2_EvictControl</td>
</tr>
<tr>
<td>31</td>
<td>Clocks and Timers</td>
</tr>
<tr>
<td>31.1</td>
<td>TPM2_ReadClock</td>
</tr>
<tr>
<td>31.2</td>
<td>TPM2_ClockSet</td>
</tr>
<tr>
<td>31.3</td>
<td>TPM2_ClockRateAdjust</td>
</tr>
<tr>
<td>32</td>
<td>Capability Commands</td>
</tr>
<tr>
<td>32.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>32.2</td>
<td>TPM2_GetCapability</td>
</tr>
<tr>
<td>32.3</td>
<td>TPM2_TestParms</td>
</tr>
<tr>
<td>33</td>
<td>Non-volatile Storage</td>
</tr>
<tr>
<td>33.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>33.2</td>
<td>NV Counters</td>
</tr>
<tr>
<td>33.3</td>
<td>TPM2_NV_DefineSpace</td>
</tr>
<tr>
<td>33.4</td>
<td>TPM2_NV.UndefSpace</td>
</tr>
<tr>
<td>33.5</td>
<td>TPM2_NV.UndefSpaceSpecial</td>
</tr>
<tr>
<td>33.6</td>
<td>TPM2_NV_ReadPublic</td>
</tr>
<tr>
<td>33.7</td>
<td>TPM2_NV_Write</td>
</tr>
<tr>
<td>33.8</td>
<td>TPM2_NV_Increment</td>
</tr>
<tr>
<td>33.9</td>
<td>TPM2_NV_Extend</td>
</tr>
<tr>
<td>33.10</td>
<td>TPM2_NV_SetBits</td>
</tr>
<tr>
<td>33.11</td>
<td>TPM2_NV_WriteLock</td>
</tr>
</tbody>
</table>
33.12 TPM2_NV_GlobalWriteLock ........................................................................................................ 429
33.13 TPM2_NV_Read ......................................................................................................................... 432
33.14 TPM2_NV_ReadLock ................................................................................................................ 435
33.15 TPM2_NV_ChangeAuth ............................................................................................................ 438
33.16 TPM2_NV_Certify ..................................................................................................................... 441
Part 3: Commands

Tables
Table 1 — Command Modifiers and Decoration.............................................................. 2
Table 2 — Separators .................................................................................................. 3
Table 3 — Unmarshaling Errors .................................................................................. 10
Table 4 — Command-Independent Response Codes................................................... 13
Table 5 — TPM2_Startup Command........................................................................... 21
Table 6 — TPM2_Startup Response .......................................................................... 21
Table 7 — TPM2_Shutdown Command ....................................................................... 25
Table 8 — TPM2_Shutdown Response ....................................................................... 25
Table 9 — TPM2_SelfTest Command .......................................................................... 30
Table 10 — TPM2_SelfTest Response ......................................................................... 30
Table 11 — TPM2_IncrementalSelfTest Command .................................................... 33
Table 12 — TPM2_IncrementalSelfTest Response ...................................................... 33
Table 13 — TPM2_GetTestResult Command ............................................................... 36
Table 14 — TPM2_GetTestResult Response ............................................................... 36
Table 15 — TPM2_StartAuthSession Command ......................................................... 40
Table 16 — TPM2_StartAuthSession Response ......................................................... 40
Table 17 — TPM2_PolicyRestart Command ............................................................... 44
Table 18 — TPM2_PolicyRestart Response ............................................................... 44
Table 19 — TPM2_Create Command ......................................................................... 48
Table 20 — TPM2_Create Response .......................................................................... 48
Table 21 — TPM2_Load Command ............................................................................ 52
Table 22 — TPM2_Load Response ............................................................................. 52
Table 23 — TPM2_LoadExternal Command ............................................................... 57
Table 24 — TPM2_LoadExternal Response ............................................................... 57
Table 25 — TPM2_ReadPublic Command ................................................................. 61
Table 26 — TPM2_ReadPublic Response ................................................................. 61
Table 27 — TPM2_ActivateCredential Command ..................................................... 64
Table 28 — TPM2_ActivateCredential Response ....................................................... 64
Table 29 — TPM2_MakeCredential Command .......................................................... 68
Table 30 — TPM2_MakeCredential Response ........................................................... 68
Table 31 — TPM2_Unseal Command ......................................................................... 71
Table 32 — TPM2_Unseal Response .......................................................................... 71
Table 33 — TPM2_ObjectChangeAuth Command ...................................................... 74
Table 34 — TPM2_ObjectChangeAuth Response ....................................................... 74
Table 35 — TPM2_Duplicate Command .................................................................... 78
Table 36 — TPM2_Duplicate Response ..................................................................... 78
Table 37 — TPM2_Rewrap Command ....................................................................... 82
Table 38 — TPM2_Rewrap Response ....................................................................... 82
Table 39 — TPM2_Import Command ................................................................. 88
Table 40 — TPM2_Import Response ................................................................. 88
Table 41 — Padding Scheme Selection ............................................................. 92
Table 42 — Message Size Limits Based on Padding ........................................ 93
Table 43 — TPM2_RSA_Encrypt Command ..................................................... 94
Table 44 — TPM2_RSA_Encrypt Response ...................................................... 94
Table 45 — TPM2_RSA_Decrypt Command ..................................................... 98
Table 46 — TPM2_RSA_Decrypt Response ...................................................... 98
Table 47 — TPM2_ECDH_KeyGen Command .................................................. 102
Table 48 — TPM2_ECDH_KeyGen Response .................................................. 102
Table 49 — TPM2_ECDH_ZGen Command ..................................................... 105
Table 50 — TPM2_ECDH_ZGen Response ..................................................... 105
Table 51 — TPM2_ECC_Parameters Command ............................................. 107
Table 52 — TPM2_ECC_Parameters Response .............................................. 107
Table 53 — TPM2_ZGen_2Phase Command .................................................. 110
Table 54 — TPM2_ZGen_2Phase Response .................................................. 110
Table 55 — Symmetric Chaining Process ....................................................... 114
Table 56 — TPM2_EncryptDecrypt Command .............................................. 116
Table 57 — TPM2_EncryptDecrypt Response .............................................. 116
Table 58 — TPM2_Hash Command ............................................................... 120
Table 59 — TPM2_Hash Response ............................................................... 120
Table 60 — TPM2_HMAC Command ............................................................. 123
Table 61 — TPM2_HMAC Response .............................................................. 123
Table 62 — TPM2_GetRandom Command ..................................................... 127
Table 63 — TPM2_GetRandom Response ....................................................... 127
Table 64 — TPM2_StirRandom Command .................................................... 130
Table 65 — TPM2_StirRandom Response ...................................................... 130
Table 66 — Hash Selection Matrix ............................................................... 132
Table 67 — TPM2_HMAC_Start Command .................................................. 133
Table 68 — TPM2_HMAC_Start Response .................................................... 133
Table 69 — TPM2_HashSequenceStart Command ....................................... 137
Table 70 — TPM2_HashSequenceStart Response ....................................... 137
Table 71 — TPM2_SequenceUpdate Command ........................................... 140
Table 72 — TPM2_SequenceUpdate Response ........................................... 140
Table 73 — TPM2_SequenceComplete Command ....................................... 144
Table 74 — TPM2_SequenceComplete Response ........................................ 144
Table 75 — TPM2_EventSequenceComplete Command ................................ 148
Table 76 — TPM2_EventSequenceComplete Response .................................. 148
Table 77 — TPM2_Certify Command ............................................................ 154
Part 3: Commands

Table 116

Table 115

Table 114

Table 113

Table 112

Table 111

Table 110

Table 109

Table 108

Table 107

Table 106

Table 105

Table 104

Table 103

Table 102

Table 101

Table 100

Table 99

Table 98

Table 97

Table 96

Table 95

Table 94

Table 93

Table 92

Table 91

Table 90

Table 89

Table 88

Table 87

Table 86

Table 85

Table 84

Table 83

Table 82

Table 81

Table 80

Table 79

Table 78

Table 77

Table 76

Table 75

Table 74

Table 73

Table 72

Table 71

Table 70

Table 69

Table 68

Table 67

Table 66

Table 65

Table 64

Table 63

Table 62

Table 61

Table 60

Table 59

Table 58

Table 57

Table 56

Table 55

Table 54

Table 53

Table 52

Table 51

Table 50

Table 49

Table 48

Table 47

Table 46

Table 45

Table 44

Table 43

Table 42

Table 41

Table 40

Table 39

Table 38

Table 37

Table 36

Table 35

Table 34

Table 33

Table 32

Table 31

Table 30

Table 29

Table 28

Table 27

Table 26

Table 25

Table 24

Table 23

Table 22

Table 21

Table 20

Table 19

Table 18

Table 17

Table 16

Table 15

Table 14

Table 13

Table 12

Table 11

Table 10

Table 9

Table 8

Table 7

Table 6

Table 5

Table 4

Table 3

Table 2

Table 1

Table 78 — TPM2_Certify Response ........................................................................154
Table 79 — TPM2_CertifyCreation Command ...........................................................158
Table 80 — TPM2_CertifyCreation Response ..............................................................158
Table 81 — TPM2_Quote Command ............................................................................162
Table 82 — TPM2_Quote Response .............................................................................162
Table 83 — TPM2_GetSessionAuditDigest Command .................................................166
Table 84 — TPM2_GetSessionAuditDigest Response ..................................................166
Table 85 — TPM2_GetCommandAuditDigest Command .............................................170
Table 86 — TPM2_GetCommandAuditDigest Response ..............................................170
Table 87 — TPM2_GetTime Command .......................................................................174
Table 88 — TPM2_GetTime Response ........................................................................174
Table 89 — TPM2_Commit Command .......................................................................180
Table 90 — TPM2_Commit Response .........................................................................180
Table 91 — TPM2_EC_Ephemeral Command ..............................................................185
Table 92 — TPM2_EC_Ephemeral Response .................................................................185
Table 93 — TPM2_VerifySignature Command ............................................................188
Table 94 — TPM2_VerifySignature Response .............................................................188
Table 95 — TPM2_Sign Command .............................................................................192
Table 96 — TPM2_Sign Response ................................................................................192
Table 97 — TPM2_SetCommandCodeAuditStatus Command ....................................197
Table 98 — TPM2_SetCommandCodeAuditStatus Response .....................................197
Table 99 — TPM2_PCR_Extend Command ................................................................202
Table 100 — TPM2_PCR_Extend Response .................................................................202
Table 101 — TPM2_PCR_Event Command ................................................................205
Table 102 — TPM2_PCR_Event Response ................................................................205
Table 103 — TPM2_PCR_Read Command ................................................................208
Table 104 — TPM2_PCR_Read Response ................................................................208
Table 105 — TPM2_PCR_Allocate Command .............................................................211
Table 106 — TPM2_PCR_Allocate Response ...............................................................211
Table 107 — TPM2_PCR_SetAuthPolicy Command ....................................................214
Table 108 — TPM2_PCR_SetAuthPolicy Response .....................................................214
Table 109 — TPM2_PCR_SetAuthValue Command ....................................................217
Table 110 — TPM2_PCR_SetAuthValue Response .....................................................217
Table 111 — TPM2_PCR_Reset Command ................................................................220
Table 112 — TPM2_PCR_Reset Response ................................................................220
Table 113 — TPM2_PolicySigned Command ..............................................................236
Table 114 — TPM2_PolicySigned Response ...............................................................236
Table 115 — TPM2_PolicySecret Command ..............................................................241
Table 116 — TPM2_PolicySecret Response ...............................................................241
Table 117 — TPM2_PolicyTicket Command ................................................................. 245
Table 118 — TPM2_PolicyTicket Response ............................................................... 245
Table 119 — TPM2_PolicyOR Command ................................................................... 249
Table 120 — TPM2_PolicyOR Response ................................................................. 249
Table 121 — TPM2_PolicyPCR Command ............................................................... 253
Table 122 — TPM2_PolicyPCR Response ............................................................... 253
Table 123 — TPM2_PolicyLocality Command ......................................................... 257
Table 124 — TPM2_PolicyLocality Response ......................................................... 257
Table 125 — TPM2_PolicyNV Command ............................................................... 261
Table 126 — TPM2_PolicyNV Response ............................................................... 261
Table 127 — TPM2_PolicyCounterTimer Command ............................................... 266
Table 128 — TPM2_PolicyCounterTimer Response ............................................... 266
Table 129 — TPM2_PolicyCommandCode Command ............................................. 271
Table 130 — TPM2_PolicyCommandCode Response .............................................. 271
Table 131 — TPM2_PolicyPhysicalPresence Command ......................................... 274
Table 132 — TPM2_PolicyPhysicalPresence Response .......................................... 274
Table 133 — TPM2_PolicyCpHash Command ......................................................... 277
Table 134 — TPM2_PolicyCpHash Response ......................................................... 277
Table 135 — TPM2_PolicyNameHash Command .................................................... 281
Table 136 — TPM2_PolicyNameHash Response ....................................................... 281
Table 137 — TPM2_PolicyDuplicationSelect Command ......................................... 284
Table 138 — TPM2_PolicyDuplicationSelect Response ......................................... 284
Table 139 — TPM2_PolicyAuthorize Command ..................................................... 288
Table 140 — TPM2_PolicyAuthorize Response ...................................................... 288
Table 141 — TPM2_PolicyAuthValue Command ..................................................... 292
Table 142 — TPM2_PolicyAuthValue Response ...................................................... 292
Table 143 — TPM2_PolicyPassword Command ...................................................... 295
Table 144 — TPM2_PolicyPassword Response ....................................................... 295
Table 145 — TPM2_PolicyGetDigest Command ..................................................... 298
Table 146 — TPM2_PolicyGetDigest Response ....................................................... 298
Table 133 — TPM2_PolicyNvWritten Command ..................................................... 301
Table 134 — TPM2_PolicyNvWritten Response ...................................................... 301
Table 147 — TPM2_CreatePrimary Command ....................................................... 305
Table 148 — TPM2_CreatePrimary Response ......................................................... 305
Table 149 — TPM2_HierarchyControl Command ................................................. 309
Table 150 — TPM2_HierarchyControl Response .................................................... 309
Table 151 — TPM2_SetPrimaryPolicy Command ................................................... 313
Table 152 — TPM2_SetPrimaryPolicy Response .................................................... 313
Table 153 — TPM2_ChangePPS Command ............................................................. 316
TPM2_GetCapability Response
TPM2_GetCapability Command
TPM2_ClockRateAdjust Response
TPM2_ClockRateAdjust Command
TPM2_ClockSet Response
TPM2_ClockSet Command
TPM2_ReadClock Command
TPM2_EvictControl Response
TPM2_EvictControl Command
TPM2_FlushContext Command
TPM2_ContextSave Response
TPM2_ContextSave Command
TPM2_FirmwareRead Response
TPM2_FirmwareRead Command
TPM2_FieldUpgradeData Response
TPM2_FieldUpgradeData Command
TPM2_FieldUpgradeStart Command
TPM2_SetAlgorithmSet Response
TPM2_SetAlgorithmSet Command
TPM2_PP_Commands Command
TPM2_DictionaryAttackParameters Command
TPM2_HierarchyChangeAuth Response
TPM2_HierarchyChangeAuth Command
TPM2_ClearControl Response
TPM2_ClearControl Command
TPM2_Clear Command
TPM2_ChangeEPS Command
TPM2_ChangePPS Response
Table 193 — TPM2_TestParms Command ................................................................. 390
Table 194 — TPM2_TestParms Response ............................................................... 390
Table 195 — TPM2_NV_DefineSpace Command ...................................................... 396
Table 196 — TPM2_NV_DefineSpace Response ....................................................... 396
Table 197 — TPM2_NV.UndefineSpace Command ................................................... 401
Table 198 — TPM2_NV.UndefineSpace Response .................................................... 401
Table 199 — TPM2_NV.UndefineSpaceSpecial Command ....................................... 404
Table 200 — TPM2_NV.UndefineSpaceSpecial Response ....................................... 404
Table 201 — TPM2_NV_ReadPublic Command ...................................................... 407
Table 202 — TPM2_NV_ReadPublic Response ....................................................... 407
Table 203 — TPM2_NV_Write Command ............................................................... 410
Table 204 — TPM2_NV_Write Response ............................................................... 410
Table 205 — TPM2_NV_Increment Command ....................................................... 414
Table 206 — TPM2_NV_Increment Response ......................................................... 414
Table 207 — TPM2_NV_Extend Command ............................................................. 418
Table 208 — TPM2_NV_Extend Response ............................................................. 418
Table 209 — TPM2_NV_SetBits Command ............................................................ 422
Table 210 — TPM2_NV_SetBits Response ............................................................. 422
Table 211 — TPM2_NV_WriteLock Command ....................................................... 426
Table 212 — TPM2_NV_WriteLock Response ....................................................... 426
Table 213 — TPM2_NV_GlobalWriteLock Command .......................................... 430
Table 214 — TPM2_NV_GlobalWriteLock Response ............................................. 430
Table 215 — TPM2_NV_Read Command ............................................................... 433
Table 216 — TPM2_NV_Read Response ............................................................... 433
Table 217 — TPM2_NV_ReadLock Command ....................................................... 436
Table 218 — TPM2_NV_ReadLock Response ....................................................... 436
Table 219 — TPM2_NV_ChangeAuth Command .................................................... 439
Table 220 — TPM2_NV_ChangeAuth Response .................................................... 439
Table 221 — TPM2_NV_Certify Command ........................................................... 442
Table 222 — TPM2_NV_Certify Response ........................................................... 442
1 Scope

This part 3 of the Trusted Module Library specification contains the definitions of the TPM commands. These commands make use of the constants, flags, structure, and union definitions defined in part 2: Structures.

The detailed description of the operation of the commands is written in the C language with extensive comments. The behavior of the C code in this part 3 is normative but does not fully describe the behavior of a TPM. The combination of this part 3 and part 4: Supporting Routines is sufficient to fully describe the required behavior of a TPM.

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 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 part 1 of this specification apply.

3 Symbols and abbreviated terms

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

4 Notation

4.1 Introduction

In addition to the notation in this clause, the “Notations” clause in Part 1 of this specification is applicable to this Part 3.

Command and response tables used various decorations to indicate the fields of the command and the allowed types. These decorations are described in this clause.

4.2 Table Decorations

The symbols and terms in the Notation column of Table 1 are used in the tables for the command schematics. These values indicate various qualifiers for the parameters or descriptions with which they are associated.
### Table 1 — Command Modifiers and Decoration

<table>
<thead>
<tr>
<th>Notation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>A Type decoration – When appended to a value in the Type column of a command, this symbol indicates that the parameter is allowed to use the “null” value of the data type (see &quot;Conditional Types&quot; in Part 2). The null value is usually TPM_RH_NULL for a handle or TPM_ALG_NULL for an algorithm selector.</td>
</tr>
<tr>
<td>@</td>
<td>A Name decoration – When this symbol precedes a handle parameter in the “Name” column, it indicates that an authorization session is required for use of the entity associated with the handle. If a handle does not have this symbol, then an authorization session is not allowed.</td>
</tr>
<tr>
<td>+PP</td>
<td>A Description modifier – This modifier may follow TPM_RH_PLATFORM in the “Description” column to indicate that Physical Presence is required when platformAuth/platformPolicy is provided.</td>
</tr>
<tr>
<td>+(PP)</td>
<td>A Description modifier – This modifier may follow TPM_RH_PLATFORM to indicate that Physical Presence may be required when platformAuth/platformPolicy is provided. The commands with this notation may be in the setList or clearList of TPM2_PP_Commands().</td>
</tr>
</tbody>
</table>
| {NV}     | A Description modifier – This modifier may follow the commandCode in the “Description” column to indicate that the command may result in an update of NV memory and be subject to rate throttling by the TPM. If the command code does not have this notation, then a write to NV memory does not occur as part of the command actions.  
NOTE Any command that uses authorization may cause a write to NV if there is an authorization failure. A TPM may use the occasion of command execution to update the NV copy of clock. |
| (F)      | A Description modifier – This modifier indicates that the “flushed” attribute will be SET in the TPMA_CC for the command. The modifier may follow the commandCode in the “Description” column to indicate that any transient handle context used by the command will be flushed from the TPM when the command completes. This may be combined with the (NV) modifier but not with the (E) modifier.  
EXAMPLE 1  {NV F}  
EXAMPLE 2  TPM2_SequenceComplete() will flush the context associated with the sequenceHandle. |
| (E)      | A Description modifier – This modifier indicates that the “extensive” attribute will be SET in the TPMA_CC for the command. This modifier may follow the commandCode in the “Description” column to indicate that the command may flush many objects and re-enumeration of the loaded context likely will be required. This may be combined with the (NV) modifier but not with the (F) modifier.  
EXAMPLE 1  {NV E}  
EXAMPLE 2  TPM2_Clear() will flush all contexts associated with the Storage hierarchy and the Endorsement hierarchy. |
| Auth Index: | A Description modifier – When a handle has a “@” decoration, the “Description” column will contain an “Auth Index:” entry for the handle. This entry indicates the number of the authorization session. The authorization sessions associated with handles will occur in the session area in the order of the handles with the “@” modifier. Sessions used only for encryption/decryption or only for audit will follow the handles used for authorization. |
Notation | Meaning
---|---
Auth Role: A Description modifier – This will be in the “Description” column of a handle with the “@” decoration. It may have a value of USER, ADMIN or DUP. If the handle has the Auth Role of USER and the handle is an Object, the type of authorization is determined by the setting of userWithAuth in the Object’s attributes. If the Auth Role is ADMIN and the handle is an Object, the type of authorization is determined by the setting of adminWithPolicy in the Object’s attributes. If the DUP role is selected, authorization may only be with a policy session (DUP role only applies to Objects). When either ADMIN or DUP role is selected, a policy command that selects the command being authorized is required to be part of the policy.

EXAMPLE: TPM2_Certify requires the ADMIN role for the first handle (objectHandle). The policy authorization for objectHandle is required to contain TPM2_PolicyCommandCode(commandCode == TPM_CC_Certify). This sets the state of the policy so that it can be used for ADMIN role authorization in TPM2_Certify().

If the handle references an NV Index, then the allowed authorizations are determined by the settings of the attributes of the NV Index as described in Part 2, "TPMA_NV (NV Index Attributes)."

### 4.3 Handle and Parameter Demarcation

The demarcations between the header, handle, and parameter parts are indicated by:

<table>
<thead>
<tr>
<th>Separator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>the values immediately following are in the handle area</td>
</tr>
<tr>
<td></td>
<td>the values immediately following are in the parameter area</td>
</tr>
</tbody>
</table>

### 4.4 AuthorizationSize and ParameterSize

Authorization sessions are not shown in the command or response schematics. When the tag of a command or response is TPM_ST_SESSIONS, then a 32-bit value will be present in the command/response buffer to indicate the size of the authorization field or the parameter field. This value shall immediately follow the handle area (which may contain no handles). For a command, this value (authorizationSize) indicates the size of the Authorization Area and shall have a value of 9 or more. For a response, this value (parameterSize) indicates the size of the parameter area and may have a value of zero.

If the authorizationSize field is present in the command, parameterSize will be present in the response, but only if the responseCode is TPM_RC_SUCCESS.

When the command tag is TPM_ST_NO_SESSIONS, no authorizations are present and no authorizationSize field is required and shall not be present.
5 Normative References

The “Normative References” clause in Part 1 of this specification is applicable to this Part 3.

6 Symbols and Abbreviated Terms

The “Symbols and Abbreviated Terms” clause in Part 1 of this specification is applicable to this Part 3.

7 Command Processing

7.1 Introduction

This clause defines the command validations that are required of any implementation and the response code returned if the indicated check fails. Unless stated otherwise, the order of the checks is not normative and different TPM may give different responses when a command has multiple errors.

In the description below, some statements that describe a check may be followed by a response code in parentheses. This is the normative response code should the indicated check fail. A normative response code may also be included in the statement.

7.2 Command Header Validation

Before a TPM may begin the actions associated with a command, a set of command format and consistency checks shall be performed. These checks are listed below and should be performed in the indicated order.

a) The TPM shall successfully unmarshal a TPMI_ST_COMMAND_TAG and verify that it is either TPM_ST_SESSIONS or TPM_ST_NO_SESSIONS (TPM_RC_BAD_TAG).

b) The TPM shall successfully unmarshal a UINT32 as the commandSize. If the TPM has an interface buffer that is loaded by some hardware process, the number of octets in the input buffer for the command reported by the hardware process shall exactly match the value in commandSize (TPM_RC_COMMAND_SIZE).

NOTE A TPM may have direct access to system memory and unmarshal directly from that memory.

c) The TPM shall successfully unmarshal a TPM_CC and verify that the command is implemented (TPM_RC_COMMAND_CODE).

7.3 Mode Checks

The following mode checks shall be performed in the order listed:
a) If the TPM is in Failure mode, then the commandCode is TPM_CC_GetTestResult or TPM_CC_GetCapability (TPM_RC_FAILURE) and the command tag is TPM_ST_NO_SESSIONS (TPM_RC_FAILURE).

NOTE 1 In Failure mode, the TPM has no cryptographic capability and processing of sessions is not supported.

b) The TPM is in Field Upgrade mode (FUM), the commandCode is TPM_CC_FieldUpgradeData (TPM_RC_UPGRADE).

c) If the TPM has not been initialized (TPM2_Startup()), then the commandCode is TPM_CC_Startup (TPM_RC_INITIALIZE).

NOTE 2 The TPM may enter Failure mode during _TPM_Init processing, before TPM2_Startup(). Since the platform firmware cannot know that the TPM is in Failure mode without accessing it, and since the first command is required to be TPM2_Startup(), the expected sequence will be that platform firmware (the CRTM) will issue TPM2_Startup() and receive TPM_RC_FAILURE indicating that the TPM is in Failure mode.

There may be failures where a TPM cannot record that it received TPM2_Startup(). In those cases, a TPM in failure mode may process TPM2_GetTestResult(), TPM2_GetCapability(), or the field upgrade commands. As a side effect, that TPM may process TPM2_GetTestResult(), TPM2_GetCapability() or the field upgrade commands before TPM2_Startup().

This is a corner case exception to the rule that TPM2_Startup() must be the first command.

The mode checks may be performed before or after the command header validation.

7.4 Handle Area Validation

After successfully unmarshaling and validating the command header, the TPM shall perform the following checks on the handles and sessions. These checks may be performed in any order.

a) The TPM shall successfully unmarshal the number of handles required by the command and validate that the value of the handle is consistent with the command syntax. If not, the TPM shall return TPM_RC_VALUE.

NOTE 1 The TPM may unmarshal a handle and validate that it references an entity on the TPM before unmarshaling a subsequent handle.

NOTE 2 If the submitted command contains fewer handles than required by the syntax of the command, the TPM may continue to read into the next area and attempt to interpret the data as a handle.

b) For all handles in the handle area of the command, the TPM will validate that the referenced entity is present in the TPM.

1) If the handle references a transient object, the handle shall reference a loaded object (TPM_RC_REFERENCE_H0 + N where N is the number of the handle in the command).

NOTE 3 If the hierarchy for a transient object is disabled, then the transient objects will be flushed so this check will fail.

2) If the handle references a persistent object, then
   i) the handle shall reference a persistent object that is currently in TPM non-volatile memory (TPM_RC_HANDLE);
   ii) the hierarchy associated with the object is not disabled (TPM_RC_HIERARCHY); and
   iii) if the TPM implementation moves a persistent object to RAM for command processing then sufficient RAM space is available (TPM_RC_OBJECT_MEMORY).
3) If the handle references an NV Index, then
   i) an Index exists that corresponds to the handle (TPM_RC_HANDLE); and
   ii) the hierarchy associated with the existing NV Index is not disabled (TPM_RC_HANDLE).
   iii) the hierarchy associated with an NV index being defined is not disabled (TPM_RC_HIERARCHY)
4) If the handle references a session, then the session context shall be present in TPM memory (TPM_RC_REFERENCE_S0 + N).
5) If the handle references a primary seed for a hierarchy (TPM_RH_ENDORSEMENT, TPM_RH_OWNER, or TPM_RH_PLATFORM) then the enable for the hierarchy is SET (TPM_RC_HIERARCHY).
6) If the handle references a PCR, then the value is within the range of PCR supported by the TPM (TPM_RC_VALUE)

   **NOTE 4** In the reference implementation, this TPM_RC_VALUE is returned by the unmarshaling code for a TPMI_DH_PCR.

### 7.5 Session Area Validation

a) If the tag is TPM_ST_SESSIONS and the command is a context management command (TPM2_ContextSave(), TPM2_ContextLoad(), or TPM2_FlushContext()) the TPM will return TPM_RC_AUTH_CONTEXT.
b) If the tag is TPM_ST_SESSIONS, the TPM will attempt to unmarshal an authorizationSize and return TPM_RC_AUTHSIZE if the value is not within an acceptable range.
   1) The minimum value is (sizeof(TPM_HANDLE) + sizeof(UINT16) + sizeof(TPMA_SESSION) + sizeof(UINT16)).
   2) The maximum value of authorizationSize is equal to commandSize – (sizeof(TPM_ST) + sizeof(UINT32) + sizeof(TPM_CC) + (N * sizeof(TPM_HANDLE)) + sizeof(UINT32)) where N is the number of handles associated with the commandCode and may be zero.

   **NOTE 1** (sizeof(TPM_ST) + sizeof(UINT32) + sizeof(TPM_CC)) is the size of a command header. The last UINT32 contains the authorizationSize octets, which are not counted as being in the authorization session area.

c) The TPM will unmarshal the authorization sessions and perform the following validations:
   1) If the session handle is not a handle for an HMAC session, a handle for a policy session, or, TPM_RS_PW then the TPM shall return TPM_RC_HANDLE.
   2) If the session is not loaded, the TPM will return the warning TPM_RC_REFERENCE_S0 + N where N is the number of the session. The first session is session zero, N = 0.

   **NOTE 2** If the HMAC and policy session contexts use the same memory, the type of the context must match the type of the handle.

   3) If the maximum allowed number of sessions have been unmarshaled and fewer octets than indicated in authorizationSize were unmarshaled (that is, authorizationSize is too large), the TPM shall return TPM_RC_AUTHSIZE.
4) The consistency of the authorization session attributes is checked.
   
   i) An authorization session is present for each of the handles with the "@" decoration (TPM_RC_AUTH_MISSING).
   
   ii) Only one session is allowed for:
       
       a) session auditing (TPM_RC_ATTRIBUTES) – this session may be used for encrypt or decrypt but may not be a session that is also used for authorization;
       
       b) decrypting a command parameter (TPM_RC_ATTRIBUTES) – this may be any of the authorization sessions, or the audit session, or a session may be added for the single purpose of decrypting a command parameter, as long as the total number of sessions does not exceed three; and
       
       c) encrypting a response parameter (TPM_RC_ATTRIBUTES) – this may be any of the authorization sessions, or the audit session if present, or a session may be added for the single purpose of encrypting a response parameter, as long as the total number of sessions does not exceed three.

   NOTE 3 A session used for decrypting a command parameter may also be used for encrypting a response parameter.

7.6 Authorization Checks

After unmarshaling and validating the handles and the consistency of the authorization sessions, the authorizations shall be checked. Authorization checks only apply to handles if the handle in the command schematic has the "@" decoration.

a) The public and sensitive portions of the object shall be present on the TPM (TPM_RC_AUTH_UNAVAILABLE).

b) If the associated handle is TPM_RH_PLATFORM, and the command requires confirmation with physical presence, then physical presence is asserted (TPM_RC_PP).

c) If the object or NV Index is subject to DA protection, and the authorization is with an HMAC or password, then the TPM is not in lockout (TPM_RC_LOCKOUT).

   NOTE 1 An object is subject to DA protection if its noDA attribute is CLEAR. An NV Index is subject to DA protection if its TPMA_NV_NO_DA attribute is CLEAR.

   NOTE 2 An HMAC or password is required in a policy session when the policy contains TPM2_PolicyAuthValue() or TPM2_PolicyPassword().

d) If the command requires a handle to have DUP role authorization, then the associated authorization session is a policy session (TPM_RC_POLICY_FAIL).

e) If the command requires a handle to have ADMIN role authorization:

   1) If the entity being authorized is an object and its adminWithPolicy attribute is SET, then the authorization session is a policy session (TPM_RC_POLICY_FAIL).

      NOTE 3 If adminWithPolicy is CLEAR, then any type of authorization session is allowed.

   2) If the entity being authorized is an NV Index, then the associated authorization session is a policy session.

      NOTE 4 The only commands that are currently defined that required use of ADMIN role authorization are commands that operate on objects and NV Indices.
f) If the command requires a handle to have USER role authorization:
   1) If the entity being authorized is an object and its userData attribute is CLEAR, then the
      associated authorization session is a policy session (TPM_RC_POLICY_FAIL).
   2) If the entity being authorized is an NV Index;
      i) if the authorization session is a policy session;
         (a) the TPMA_NV_POLICYWRITE attribute of the NV Index is SET if the command modifies
             the NV Index data (TPM_RC_AUTH_UNAVAILABLE);
         (b) the TPMA_NV_POLICYREAD attribute of the NV Index is SET if the command reads the
             NV Index data (TPM_RC_AUTH_UNAVAILABLE);
      ii) if the authorization is an HMAC session or a password;
         (a) the TPMA_NV_AUTHWRITE attribute of the NV Index is SET if the command modifies
             the NV Index data (TPM_RC_AUTH_UNAVAILABLE);
         (b) the TPMA_NV_AUTHREAD attribute of the NV Index is SET if the command reads the
             NV Index data (TPM_RC_AUTH_UNAVAILABLE).

   g) If the authorization is provided by a policy session, then:
      1) if policySession→timeOut has been set, the session shall not have expired
         (TPM_RC_EXPIRED);
      2) if policySession→cpHash has been set, it shall match the cpHash of the command
         (TPM_RC_POLICY_FAIL);
      3) if policySession→commandCode has been set, then commandCode of the command shall match
         (TPM_RC_POLICY_CC);
      4) policySession→policyDigest shall match the authPolicy associated with the handle
         (TPM_RC_POLICY_FAIL);  
      5) if policySession→pcrUpdateCounter has been set, then it shall match the value of
         pcrUpdateCounter (TPM_RC_PCR_CHANGED);
      6) if policySession→commandLocality has been set, it shall match the locality of the command
         (TPM_RC_LOCALITY), and
      7) if the authorization uses an HMAC, then the HMAC is properly constructed using the authValue
         associated with the handle and/or the session secret (TPM_RC_AUTH_FAIL or
         TPM_RC_BAD_AUTH).

   NOTE 5 For a bound session, if the handle references the object used to initiate the session, then the
   authValue will not be required but proof of knowledge of the session secret is necessary.

   NOTE 6 A policy session may require proof of knowledge of the authValue of the object being authorized.

If the TPM returns an error other than TPM_RC_AUTH_FAIL then the TPM shall not alter any TPM state.
If the TPM return TPM_RC_AUTH_FAIL, then the TPM shall not alter any TPM state other than lockoutCount.

   NOTE 7 The TPM may decrease failedTries regardless of any other processing performed by the TPM. That
   is, the TPM may exit Lockout mode, regardless of the return code.

7.7 Parameter Decryption

If an authorization session has the TPMA_SESSION.decrypt attribute SET, and the command does not
allow a command parameter to be encrypted, then the TPM will return TPM_RC_ATTRIBUTES.
Otherwise, the TPM will decrypt the parameter using the values associated with the session before parsing parameters.

### 7.8 Parameter Unmarshaling

#### 7.8.1 Introduction

The detailed actions for each command assume that the input parameters of the command have been unmarshaled into a command-specific structure with the structure defined by the command schematic. Additionally, a response-specific output structure is assumed which will receive the values produced by the detailed actions.

**NOTE** An implementation is not required to process parameters in this manner or to separate the parameter parsing from the command actions. This method was chosen for the specification so that the normative behavior described by the detailed actions would be clear and unencumbered.

Unmarshaling is the process of processing the parameters in the input buffer and preparing the parameters for use by the command-specific action code. No data movement need take place but it is required that the TPM validate that the parameters meet the requirements of the expected data type as defined in Part 2 of this specification.

#### 7.8.2 Unmarshaling Errors

When an error is encountered while unmarshaling a command parameter, an error response code is returned and no command processing occurs. A table defining a data type may have response codes embedded in the table to indicate the error returned when the input value does not match the parameters of the table.

**NOTE** In the reference implementation, a parameter number is added to the response code so that the offending parameter can be isolated. This is optional.

In many cases, the table contains no specific response code value and the return code will be determined as defined in Table 3.
Table 3 — Unmarshaling Errors

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ASYMMETRIC</td>
<td>a parameter that should be an asymmetric algorithm selection does not have a value that is supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_BAD_TAG</td>
<td>a parameter that should be a command tag selection has a value that is not supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_COMMAND_CODE</td>
<td>a parameter that should be a command code does not have a value that is supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_HASH</td>
<td>a parameter that should be a hash algorithm selection does not have a value that is supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_INSUFFICIENT</td>
<td>the input buffer did not contain enough octets to allow unmarshaling of the expected data type;</td>
</tr>
<tr>
<td>TPM_RC_KDF</td>
<td>a parameter that should be a key derivation scheme (KDF) selection does not have a value that is supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_KEY_SIZE</td>
<td>a parameter that is a key size has a value that is not supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_MODE</td>
<td>a parameter that should be a symmetric encryption mode selection does not have a value that is supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_RESERVED</td>
<td>a non-zero value was found in a reserved field of an attribute structure (TPMA_)</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>a parameter that should be signing or encryption scheme selection does not have a value that is supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>the value of a size parameter is larger or smaller than allowed</td>
</tr>
<tr>
<td>TPM_RC_SYMMETRIC</td>
<td>a parameter that should be a symmetric algorithm selection does not have a value that is supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_TAG</td>
<td>a parameter that should be a structure tag has a value that is not supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>The type parameter of a TPMT_PUBLIC or TPMT_SENSITIVE has a value that is not supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>a parameter does not have one of its allowed values</td>
</tr>
</tbody>
</table>

In some commands, a parameter may not be used because of various options of that command. However, the unmarshaling code is required to validate that all parameters have values that are allowed by the Part 2 definition of the parameter type even if that parameter is not used in the command actions.

7.9 Command Post Processing

When the code that implements the detailed actions of the command completes, it returns a response code. If that code is not TPM_RC_SUCCESS, the post processing code will not update any session or audit data and will return a 10-octet response packet.

If the command completes successfully, the tag of the command determines if any authorization sessions will be in the response. If so, the TPM will encrypt the first parameter of the response if indicated by the authorization attributes. The TPM will then generate a new nonce value for each session and, if appropriate, generate an HMAC.
NOTE 1 The authorization attributes were validated during the session area validation to ensure that only one session was used for parameter encryption of the response and that the command allowed encryption in the response.

NOTE 2 No session nonce value is used for a password authorization but the session data is present.

Additionally, if the command is being audited by Command Audit, the audit digest is updated with the $cpHash$ of the command and $rpHash$ of the response.
8 Response Values

8.1 Tag

When a command completes successfully, the tag parameter in the response shall have the same value as the tag parameter in the command (TPM_ST_SESSIONS or TPM_RC_NO_SESSIONS). When a command fails (the responseCode is not TPM_RC_SUCCESS), then the tag parameter in the response shall be TPM_ST_NO_SESSIONS.

A special case exists when the command tag parameter is not an allowed value (TPM_ST_SESSIONS or TPM_ST_NO_SESSIONS). For this case, it is assumed that the system software is attempting to send a command formatted for a TPM 1.2 but the TPM is not capable of executing TPM 1.2 commands. So that the TPM 1.2 compatible software will have a recognizable response, the TPM sets tag to TPM_ST_RSP_COMMAND, responseSize to 00 00 00 0A16 and responseCode to TPM_RC_BAD_TAG. This is the same response as the TPM 1.2 fatal error for TPM_BADTAG.

8.2 Response Codes

The normal response for any command is TPM_RC_SUCCESS. Any other value indicates that the command did not complete and the state of the TPM is unchanged. An exception to this general rule is that the logic associated with dictionary attack protection is allowed to be modified when an authorization failure occurs.

Commands have response codes that are specific to that command, and those response codes are enumerated in the detailed actions of each command. The codes associated with the unmarshaling of parameters are documented Table 3. Another set of response code value are not command specific and indicate a problem that is not specific to the command. That is, if the indicated problem is remedied, the same command could be resubmitted and may complete normally.

The response codes that are not command specific are listed and described in Table 4.

The reference code for the command actions may have code that generates specific response codes associated with a specific check but the listing of responses may not have that response code listed.
## Table 4 — Command-Independent Response Codes

<table>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CANCELLLED</td>
<td>This response code may be returned by a TPM that supports command cancel. When the TPM receives an indication that the current command should be cancelled, the TPM may complete the command or return this code. If this code is returned, then the TPM state is not changed and the same command may be retried.</td>
</tr>
<tr>
<td>TPM_RC_CONTEXT_GAP</td>
<td>This response code can be returned for commands that manage session contexts. It indicates that the gap between the lowest numbered active session and the highest numbered session is at the limits of the session tracking logic. The remedy is to load the session context with the lowest number so that its tracking number can be updated.</td>
</tr>
<tr>
<td>TPM_RC_LOCKOUT</td>
<td>This response indicates that authorizations for objects subject to DA protection are not allowed at this time because the TPM is in DA lockout mode. The remedy is to wait or to execute TPM2_DictionaryAttackLockoutReset().</td>
</tr>
<tr>
<td>TPM_RC_MEMORY</td>
<td>A TPM may use a common pool of memory for objects, sessions, and other purposes. When the TPM does not have enough memory available to perform the actions of the command, it may return TPM_RC_MEMORY. This indicates that the TPM resource manager may flush either sessions or objects in order to make memory available for the command execution. A TPM may choose to return TPM_RC_OBJECT_MEMORY or TPM_RC_SESSION_MEMORY if it needs contexts of a particular type to be flushed.</td>
</tr>
<tr>
<td>TPM_RC_NV_RATE</td>
<td>This response code indicates that the TPM is rate-limiting writes to the NV memory in order to prevent wearout. This response is possible for any command that explicitly writes to NV or commands that incidentally use NV such as a command that uses authorization session that may need to update the dictionary attack logic.</td>
</tr>
<tr>
<td>TPM_RC_NV_UNAVAILABLE</td>
<td>This response code is similar to TPM_RC_NV_RATE but indicates that access to NV memory is currently not available and the command is not allowed to proceed until it is. This would occur in a system where the NV memory used by the TPM is not exclusive to the TPM and is a shared system resource.</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_HANDLES</td>
<td>This response code indicates that the TPM has exhausted its handle space and no new objects can be loaded unless the TPM is rebooted. This does not occur in the reference implementation because of the way that object handles are allocated. However, other implementations are allowed to assign each object a unique handle each time the object is loaded. A TPM using this implementation would be able to load (2^{24}) objects before the object space is exhausted.</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>This response code can be returned by any command that causes the TPM to need an object ‘slot’. The most common case where this might be returned is when an object is loaded (TPM2_Load, TPM2_CreatePrimary(), or TPM2_ContextLoad()). However, the TPM implementation is allowed to use object slots for other reasons. In the reference implementation, the TPM copies a referenced persistent object into RAM for the duration of the command. If all the slots are previously occupied, the TPM may return this value. A TPM is allowed to use object slots for other purposes and return this value. The remedy when this response is returned is for the TPM resource manager to flush a transient object.</td>
</tr>
<tr>
<td>TPM_RC_REFERENCE_Hx</td>
<td>This response code indicates that a handle in the handle area of the command is not associated with a loaded object. The value of ‘x’ is in the range 0 to 6 with a value of 0 indicating the 1st handle and 6 representing the 7th. The TPM resource manager needs to find the correct object and load it. It may then adjust the handle and retry the command.</td>
</tr>
</tbody>
</table>

**NOTE** Usually, this error indicates that the TPM resource manager has a corrupted database.
<table>
<thead>
<tr>
<th>Response Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_REFERENCE_Sx</td>
<td>This response code indicates that a handle in the session area of the command is not associated with a loaded session. The value of ‘x’ is in the range 0 to 6 with a value of 0 indicating the 1st session handle and 6 representing the 7th. The TPM resource manager needs to find the correct session and load it. It may then retry the command. NOTE Usually, this error indicates that the TPM resource manager has a corrupted database.</td>
</tr>
<tr>
<td>TPM_RC_RETRY</td>
<td>the TPM was not able to start the command</td>
</tr>
<tr>
<td>TPM_RC_SESSION_HANDLES</td>
<td>This response code indicates that the TPM does not have a handle to assign to a new session. This response is only returned by TPM2_StartAuthSession(). It is listed here because the command is not in error and the TPM resource manager can remedy the situation by flushing a session (TPM2_FlushContext()).</td>
</tr>
<tr>
<td>TPM_RC_SESSION_MEMORY</td>
<td>This response code can be returned by any command that causes the TPM to need a session ‘slot’. The most common case where this might be returned is when a session is loaded (TPM2_StartAuthSession() or TPM2_ContextLoad()). However, the TPM implementation is allowed to use object slots for other purposes. The remedy when this response is returned is for the TPM resource manager to flush a transient object.</td>
</tr>
<tr>
<td>TPM_RC_SUCCESS</td>
<td>Normal completion for any command. If the responseCode is TPM_RC_SESSIONS, then the rest of the response has the format indicated in the response schematic. Otherwise, the response is a 10 octet value indicating an error.</td>
</tr>
<tr>
<td>TPM_RC_TESTING</td>
<td>This response code indicates that the TPM is performing tests and cannot respond to the request at this time. The command may be retried.</td>
</tr>
<tr>
<td>TPM_RC_YIELDED</td>
<td>the TPM has suspended operation on the command; forward progress was made and the command may be retried. See Part 1, &quot;Multi-tasking.&quot; NOTE This cannot occur on the reference implementation.</td>
</tr>
</tbody>
</table>
9 Implementation Dependent

The actions code for each command makes assumptions about the behavior of various sub-systems. There are many possible implementations of the subsystems that would achieve equivalent results. The actions code is not written to anticipate all possible implementations of the sub-systems. Therefore, it is the responsibility of the implementer to ensure that the necessary changes are made to the actions code when the sub-system behavior changes.
10 Detailed Actions Assumptions

10.1 Introduction

The C code in the Detailed Actions for each command is written with a set of assumptions about the processing performed before the action code is called and the processing that will be done after the action code completes.

10.2 Pre-processing

Before calling the command actions code, the following actions have occurred.

- Verification that the handles in the handle area reference entities that are resident on the TPM.
  
  **NOTE** If a handle is in the parameter portion of the command, the associated entity does not have to be loaded, but the handle is required to be the correct type.

- If use of a handle requires authorization, the Password, HMAC, or Policy session associated with the handle has been verified.

- If a command parameter was encrypted using parameter encryption, it was decrypted before being unmarshaled.

- If the command uses handles or parameters, the calling stack contains a pointer to a data structure (in) that holds the unmarshaled values for the handles and commands. If the response has handles or parameters, the calling stack contains a pointer to a data structure (out) to hold the handles and parameters generated by the command.

- All parameters of the in structure have been validated and meet the requirements of the parameter type as defined in Part 2.

- Space set aside for the out structure is sufficient to hold the largest out structure that could be produced by the command.

10.3 Post Processing

When the function implementing the command actions completes,

- response parameters that require parameter encryption will be encrypted after the command actions complete;

- audit and session contexts will be updated if the command response is TPM_RC_SUCCESS; and

- the command header and command response parameters will be marshaled to the response buffer.
11 Start-up

11.1 Introduction

This clause contains the commands used to manage the startup and restart state of a TPM.

11.2 _TPM_Init

11.2.1 General Description

_TPML_Init initializes a TPM.

Initialization actions include testing code required to execute the next expected command. If the TPM is in FUM, the next expected command is TPM2_FieldUpgradeData(); otherwise, the next expected command is TPM2_Startup().

NOTE 1 If the TPM performs self-tests after receiving _TPM_Init() and the TPM enters Failure mode before receiving TPM2_Startup() or TPM2_FieldUpgradeData(), then the TPM may be able to accept TPM2_GetTestResult() or TPM2_GetCapability().

The means of signaling _TPM_Init shall be defined in the platform-specific specifications that define the physical interface to the TPM. The platform shall send this indication whenever the platform starts its boot process and only when the platform starts its boot process.

There shall be no software method of generating this indication that does not also reset the platform and begin execution of the CRTM.

NOTE 2 In the reference implementation, this signal causes an internal flag (s_initialized) to be CLEAR. While this flag is CLEAR, the TPM will only accept the next expected command described above.
11.2.2 Detailed Actions

This function is used to process a _TPM_Init() indication.

```c
#include "InternalRoutines.h"

void _TPM_Init(void)
{
    // Initialize crypto engine
    CryptInitUnits();

    // Initialize NV environment
    NvPowerOn();

    // Start clock
    TimePowerOn();

    // Set initialization state
    TPMInit();

    // Set g_DRTMHandle as unassigned
    g_DRTMHandle = TPM_RH_UNASSIGNED;

    // No H-CRTM, yet.
    g_DrtmPreStartup = FALSE;

    return;
}
```
11.3 TPM2_Startup

11.3.1 General Description

TPM2_Startup() is always preceded by _TPM_Init, which is the physical indication that TPM initialization is necessary because of a system-wide reset. TPM2_Startup() is only valid after _TPM_Init Additional TPM2_Startup() commands are not allowed after it has completed successfully. If a TPM requires TPM2_Startup() and another command is received, or if the TPM receives TPM2_Startup() when it is not required, the TPM shall return TPM_RC_INITIALIZE.

NOTE 1 See 11.2.1 for other command options for a TPM supporting field upgrade mode.

NOTE 2 _TPM_Hash_Start, _TPM_Hash_Data, and _TPM_Hash_End are not commands and a platform-specific specification may allow these indications between _TPM_Init and TPM2_Startup().

If in Failure mode the TPM shall accept TPM2_GetTestResult() and TPM2_GetCapability() even if TPM2_Startup() is not completed successfully or processed at all.

A Shutdown/Startup sequence determines the way in which the TPM will operate in response to TPM2_Startup(). The three sequences are:

1) TPM Reset – This is a Startup(CLEAR) preceded by either Shutdown(CLEAR) or no TPM2_Shutdown(). On TPM Reset, all variables go back to their default initialization state.

   NOTE 3 Only those values that are specified as having a default initialization state are changed by TPM Reset. Persistent values that have no default initialization state are not changed by this command. Values such as seeds have no default initialization state and only change due to specific commands.

2) TPM Restart – This is a Startup(CLEAR) preceded by Shutdown(STATE). This preserves much of the previous state of the TPM except that PCR and the controls associated with the Platform hierarchy are all returned to their default initialization state;

3) TPM Resume – This is a Startup(STATE) preceded by Shutdown(STATE). This preserves the previous state of the TPM including the static Root of Trust for Measurement (S-RTM) PCR and the platform controls other than the phEnable and phEnableNV.

If a TPM receives Startup(STATE) and that was not preceded by Shutdown(STATE), the TPM shall return TPM_RC_VALUE.

If, during TPM Restart or TPM Resume, the TPM fails to restore the state saved at the last Shutdown(STATE), the TPM shall enter Failure Mode and return TPM_RC.FAILURE.

On any TPM2_Startup(),

- phEnable and phEnableNV shall be SET;
- all transient contexts (objects, sessions, and sequences) shall be flushed from TPM memory;
- TPMS_TIME_INFO.time shall be reset to zero; and
- use of lockoutAuth shall be enabled if lockoutRecovery is zero.

Additional actions are performed based on the Shutdown/Startup sequence.

On TPM Reset
- **platformAuth** and **platformPolicy** shall be set to the Empty Buffer,
- tracking data for saved session contexts shall be set to its initial value,
- the object context sequence number is reset to zero,
- a new context encryption key shall be generated,
- TPMS_CLOCK_INFO.restartCount shall be reset to zero,
- TPMS_CLOCK_INFO.resetCount shall be incremented,
- the PCR Update Counter shall be clear to zero,
- **shEnable** and **ehEnable** shall be SET, and
- PCR in all banks are reset to their default initial conditions as determined by the relevant platform-specific specification.

**NOTE 4** PCR may be initialized any time between _TPM_Init and the end of TPM2_Startup(). PCR that are preserved by TPM Resume will need to be restored during TPM2_Startup().

**NOTE 5** See "Initializing PCR" in Part 1 of this specification for a description of the default initial conditions for a PCR.

**On TPM Restart**
- TPMS_CLOCK_INFO.restartCount shall be incremented,
- **shEnable** and **ehEnable** shall be SET,
- **platformAuth** and **platformPolicy** shall be set to the Empty Buffer, and
- PCR in all banks are reset to their default initial conditions.
- If a CRTM Event sequence is active, extend the PCR designated by the platform-specific specification.

**On TPM Resume**
- the H-CRTM startup method is the same for this TPM2_Startup() as for the previous TPM2_Startup(); (TPM_RC_LOCALITY)
- TPMS_CLOCK_INFO.restartCount shall be incremented; and
- PCR that are specified in a platform-specific specification to be preserved on TPM Resume are restored to their saved state and other PCR are set to their initial value as determined by a platform-specific specification.

Other TPM state may change as required to meet the needs of the implementation.

If the **startupType** is TPMS_SU_STATE and the TPM requires TPMS_SU_CLEAR, then the TPM shall return TPM_RC_VALUE.

**NOTE 6** The TPM will require TPMS_SU_CLEAR when no shutdown was performed or after Shutdown(CLEAR).

**NOTE 7** If **startupType** is neither TPMS_SU_STATE nor TPMS_SU_CLEAR, then the unmarshaling code returns TPM_RC_VALUE.
11.3.2 Command and Response

Table 5 — TPM2_Startup Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td>TPM_ST_NO_SESSIONS</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Startup (NV)</td>
</tr>
<tr>
<td>TPM_SU</td>
<td>startupType</td>
<td>TPM_SU_CLEAR or TPM_SU_STATE</td>
</tr>
</tbody>
</table>

Table 6 — TPM2_Startup Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
11.3.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Startup_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>start up type is not compatible with previous shutdown sequence</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_Startup(
    Startup_In *in // IN: input parameter list
)
{
    STARTUP_TYPE startup;
    TPM_RC result;
    BOOL prevDrtmPreStartup;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

    // Input Validation

    // Read orderly shutdown states from previous power cycle
    NvReadReserved(NV.Orderly, &g_prevOrderlyState);

    // HACK to extract the DRTM startup type associated with the previous shutdown
    prevDrtmPreStartup = (g_prevOrderlyState == (TPM_SU_STATE + 0x8000));
    if(prevDrtmPreStartup)
        g_prevOrderlyState = TPM_SU_STATE;

    // if the previous power cycle was shut down with no StateSave command, or
    // with StateSave command for CLEAR, this cycle can not startup up with
    // STATE
    if(   (   g_prevOrderlyState == SHUTDOWN_NONE
            || g_prevOrderlyState == TPM_SU_CLEAR
        )
        && in->startupType == TPM_SU_STATE
    )
        return TPM_RC_VALUE + RC_Startup_startupType;

    // Internal Date Update

    // Translate the TPM2_ShutDown and TPM2_Startup sequence into the startup
    // types.
    if(in->startupType == TPM_SU_CLEAR && g_prevOrderlyState == TPM_SU_STATE)
    {
        startup = SU_RESTART;
        // Read state reset data
        NvReadReserved(NV.STATE_RESET, &gr);
    }
    else if(in->startupType == TPM_SU_STATE && g_prevOrderlyState == TPM_SU_STATE)
    {
        // For a resume, the H-CRTM startup method must be the same
        if(g_DrtmPreStartup != prevDrtmPreStartup)
            return TPM_RC_LOCALITY;
    }
```
55 // Read state clear and state reset data
56 NvReadReserved(NV_STATE_CLEAR, &gc);
57 NvReadReserved(NV_STATE_RESET, &gr);
58 startup = SU_RESUME;
59 }
60 else
61 {
62     startup = SU_RESET;
63 }
64 // Read persistent data from NV
65 NvReadPersistent();
66 // Crypto Startup
67 CryptUtilStartup(startup);
68 // Start up subsystems
69 // Start counters and timers
70 TimeStartup(startup);
71 // Start dictionary attack subsystem
72 DAStartup(startup);
73 // Enable hierarchies
74 HierarchyStartup(startup);
75 // Restore/Initialize PCR
76 PCRStartup(startup);
77 // Restore/Initialize command audit information
78 CommandAuditStartup(startup);
79 // Object context variables
80 if(startup == SU_RESET)
81 {
82     // Reset object context ID to 0
83     gr.objectContextID = 0;
84     // Reset clearCount to 0
85     gr.clearCount= 0;
86 }
87 // Initialize object table
88 ObjectStartup();
89 // Initialize session table
90 SessionStartup(startup);
91 // Initialize index/evict data. This function clear read/write locks
92 // in NV index
93 NvEntityStartup(startup);
94 // Initialize the orderly shut down flag for this cycle to SHUTDOWN_NONE.
95 gp.orderlyState = SHUTDOWN_NONE;
96 NvWriteReserved(NV_ORDERLY, &gp.orderlyState);
97 // Update TPM internal states if command succeeded.
98 // Record a TPM2_Startup command has been received.
99 TPMRegisterStartup();
100 return TPM_RC_SUCCESS;
101 }
11.4 TPM2_Shutdown

11.4.1 General Description

This command is used to prepare the TPM for a power cycle. The shutdownType parameter indicates how the subsequent TPM2_Startup() will be processed.

For a shutdownType of any type, the volatile portion of Clock is saved to NV memory and the orderly shutdown indication is SET. NV with the TPMA_NV_ORDERY attribute will be updated.

For a shutdownType of TPM_SU_STATE, the following additional items are saved:

- tracking information for saved session contexts;
- the session context counter;
- PCR that are designated as being preserved by TPM2_Shutdown(TPM_SU_STATE);
- the PCR Update Counter;
- flags associated with supporting the TPMA_NV_WRITESTCLEAR and TPMA_NV_READSTCLEAR attributes; and
- the command audit digest and count.

The following items shall not be saved and will not be in TPM memory after the next TPM2_Startup:

- TPM-memory-resident session contexts;
- TPM-memory-resident transient objects; or
- TPM-memory-resident hash contexts created by TPM2_HashSequenceStart().

Some values may be either derived from other values or saved to NV memory.

This command saves TPM state but does not change the state other than the internal indication that the context has been saved. The TPM shall continue to accept commands. If a subsequent command changes TPM state saved by this command, then the effect of this command is nullified. The TPM MAY nullify this command for any subsequent command rather than check whether the command changed state saved by this command. If this command is nullified, and if no TPM2_Shutdown() occurs before the next TPM2_Startup(), then the next TPM2_Startup() shall be TPM2_Startup(CLEAR).
11.4.2 Command and Response

**Table 7 — TPM2_Shutdown Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Shutdown (NV)</td>
</tr>
<tr>
<td>TPM_SU</td>
<td>shutdownType</td>
<td>TPM_SU_CLEAR or TPM_SU_STATE</td>
</tr>
</tbody>
</table>

**Table 8 — TPM2_Shutdown Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
11.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Shutdown_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_TYPE</td>
<td>if PCR bank has been re-configured, a CLEAR StateSave() is required</td>
</tr>
</tbody>
</table>

TPM_RC
TPM2_Shutdown(
    Shutdown_In *in // IN: input parameter list
)
{
    TPM_RC result;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if (result != TPM_RC_SUCCESS) return result;

    // Input Validation
    if (g_pcrReConfig && in->shutdownType == TPM_SU_STATE)
        return TPM_RC_TYPE + RC_Shutdown_shutdownType;

    // Internal Data Update
    // PCR private date state save
    PCRStateSave(in->shutdownType);
    // Get DRBG state
    CryptDrbgGetPutState(GET_STATE);
    // Save all orderly data
    NvWriteReserved(NV_ORDERLY_DATA, &go);
    // Save RAM backed NV index data
    NvStateSave();
    if (in->shutdownType == TPM_SU_STATE)
        {
            // Save STATE_RESET and STATE_CLEAR data
            NvWriteReserved(NV_STATE_CLEAR, &gc);
            NvWriteReserved(NV_STATE_RESET, &gr);
        }
    else if (in->shutdownType == TPM_SU_CLEAR)
        {
            // Save STATE_RESET data
            NvWriteReserved(NV_STATE_RESET, &gr);
        }
    // Write orderly shut down state
    if (in->shutdownType == TPM_SU_CLEAR)
        gp.orderlyState = TPM_SU_CLEAR;
    else if (in->shutdownType == TPM_SU_STATE)
        gp.orderlyState = TPM_SU_STATE;
    else
```


pAssert(FALSE);
NvWriteReserved(NV_ORDERLY, &gp.orderlyState);
return TPM_RC_SUCCESS;
12 Testing

12.1 Introduction

Compliance to standards for hardware security modules may require that the TPM test its functions before the results that depend on those functions may be returned. The TPM may perform operations using testable functions before those functions have been tested as long as the TPM returns no value that depends on the correctness of the testable function.

EXAMPLE TPM2_PCR_Event() may be executed before the hash algorithms have been tested. However, until the hash algorithms have been tested, the contents of a PCR may not be used in any command if that command may result in a value being returned to the TPM user. This means that TPM2_PCR_Read() or TPM2_PolicyPCR() could not complete until the hashes have been checked but other TPM2_PCR_Event() commands may be executed even though the operation uses previous PCR values.

If a command is received that requires return of a value that depends on untested functions, the TPM shall test the required functions before completing the command.

Once the TPM has received TPM2_SelfTest() and before completion of all tests, the TPM is required to return TPM_RC_TESTING for any command that uses a function that requires a test.

If a self-test fails at any time, the TPM will enter Failure mode. While in Failure mode, the TPM will return TPM_RC_FAILURE for any command other than TPM2_GetTestResult() and TPM2_GetCapability(). The TPM will remain in Failure mode until the next _TPM_Init.
12.2 TPM2_SelfTest

12.2.1 General Description

This command causes the TPM to perform a test of its capabilities. If the fullTest is YES, the TPM will test all functions. If fullTest = NO, the TPM will only test those functions that have not previously been tested.

If any tests are required, the TPM shall either

a) return TPM_RC_TESTING and begin self-test of the required functions, or

   NOTE 1 If fullTest is NO, and all functions have been tested, the TPM shall return TPM_RC_SUCCESS.

b) perform the tests and return the test result when complete.

If the TPM uses option a), the TPM shall return TPM_RC_TESTING for any command that requires use of a testable function, even if the functions required for completion of the command have already been tested.

NOTE 2 This command may cause the TPM to continue processing after it has returned the response. So that software can be notified of the completion of the testing, the interface may include controls that would allow the TPM to generate an interrupt when the “background” processing is complete. This would be in addition to the interrupt may be available for signaling normal command completion. It is not necessary that there be two interrupts, but the interface should provide a way to indicate the nature of the interrupt (normal command or deferred command).
### 12.2.2 Command and Response

**Table 9 — TPM2_SelfTest Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_SelfTest (NV)</td>
</tr>
<tr>
<td>TPMI_YES_NO</td>
<td>fullTest</td>
<td>YES if full test to be performed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO if only test of untested functions required</td>
</tr>
</tbody>
</table>

**Table 10 — TPM2_SelfTest Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
12.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "SelfTest_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_TESTING</td>
<td>self test in process</td>
</tr>
</tbody>
</table>

TPM_RC
TPM2_SelfTest(
    SelfTest_In     *in // IN: input parameter list
) )

    // Command Output
    // Call self test function in crypt module
    return CryptSelfTest(in->fullTest);
```
12.3 TPM2_IncrementalSelfTest

12.3.1 General Description

This command causes the TPM to perform a test of the selected algorithms.

NOTE 1 The toTest list indicates the algorithms that software would like the TPM to test in anticipation of future use. This allows tests to be done so that a future command will not be delayed due to testing.

If toTest contains an algorithm that has already been tested, it will not be tested again.

NOTE 2 The only way to force retesting of an algorithm is with TPM2_SelfTest(fullTest = YES).

The TPM will return in toDoList a list of algorithms that are yet to be tested. This list is not the list of algorithms that are scheduled to be tested but the algorithms/functions that have not been tested. Only the algorithms on the toTest list are scheduled to be tested by this command.

Making toTest an empty list allows the determination of the algorithms that remain untested without triggering any testing.

If toTest is not an empty list, the TPM shall return TPM_RC_SUCCESS for this command and then return TPM_RC_TESTING for any subsequent command (including TPM2_IncrementalSelfTest()) until the requested testing is complete.

NOTE 3 If toDoList is empty, then no additional tests are required and TPM_RC_TESTING will not be returned in subsequent commands and no additional delay will occur in a command due to testing.

NOTE 4 If none of the algorithms listed in toTest is in the toDoList, then no tests will be performed.

If all the parameters in this command are valid, the TPM returns TPM_RC_SUCCESS and the toDoList (which may be empty).

NOTE 5 An implementation may perform all requested tests before returning TPM_RC_SUCCESS, or it may return TPM_RC_SUCCESS for this command and then return TPM_RC_TESTING for all subsequent commands (including TPM2_IncrementalSelfTest()) until the requested tests are complete.
12.3.2 Command and Response

Table 11 — TPM2_IncrementalSelfTest Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_IncrementalSelfTest (NV)</td>
</tr>
<tr>
<td>TPML_ALG</td>
<td>toTest</td>
<td>list of algorithms that should be tested</td>
</tr>
</tbody>
</table>

Table 12 — TPM2_IncrementalSelfTest Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPML_ALG</td>
<td>toDoList</td>
<td>list of algorithms that need testing</td>
</tr>
</tbody>
</table>
12.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "IncrementalSelfTest_fp.h"
TPM_RC
TPM2_IncrementalSelfTest(
    IncrementalSelfTest_In *in,       // IN: input parameter list
    IncrementalSelfTest_Out *out)      // OUT: output parameter list
{
    // Command Output
    // Call incremental self test function in crypt module
    return CryptIncrementalSelfTest(&in->toTest, &out->toDoList);
}
```
12.4 TPM2_GetTestResult

12.4.1 General Description

This command returns manufacturer-specific information regarding the results of a self-test and an indication of the test status.

If TPM2_SelfTest() has not been executed and a testable function has not been tested, testResult will be TPM_RC_NEEDS_TEST. If TPM2_SelfTest() has been received and the tests are not complete, testResult will be TPM_RC_TESTING. If testing of all functions is complete without functional failures, testResult will be TPM_RC_SUCCESS. If any test failed, testResult will be TPM_RC_FAILURE. If the TPM is in Failure mode because of an invalid startupType in TPM2_Startup(), testResult will be TPM_RC_INITIALIZE.

This command will operate when the TPM is in Failure mode so that software can determine the test status of the TPM and so that diagnostic information can be obtained for use in failure analysis. If the TPM is in Failure mode, then tag is required to be TPM_ST_NO_SESSIONS or the TPM shall return TPM_RC_FAILURE.
12.4.2 Command and Response

Table 13 — TPM2_GetTestResult Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_GetTestResult</td>
</tr>
</tbody>
</table>

Table 14 — TPM2_GetTestResult Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>outData</td>
<td>test result data contains manufacturer-specific information</td>
</tr>
<tr>
<td>TPM_RC</td>
<td>testResult</td>
<td></td>
</tr>
</tbody>
</table>
12.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "GetTestResult_fp.h"
TPM_RC
TPM2_GetTestResult(
    GetTestResult_Out *out // OUT: output parameter list
)
{
    // Command Output
    // Call incremental self test function in crypt module
    out->testResult = CryptGetTestResult(&out->outData);
    return TPM_RC_SUCCESS;
}
```
13 Session Commands

13.1 TPM2_StartAuthSession

13.1.1 General Description

This command is used to start an authorization session using alternative methods of establishing the session key (sessionKey). The session key is then used to derive values used for authorization and for encrypting parameters.

This command allows injection of a secret into the TPM using either asymmetric or symmetric encryption. The type of tpmKey determines how the value in encryptedSalt is encrypted. The decrypted secret value is used to compute the sessionKey.

NOTE 1 If tpmKey is TPM_RH_NULL, then encryptedSalt is required to be an Empty Buffer.

The label value of “SECRET” (see “Terms and Definitions” in Part 1 of this specification) is used in the recovery of the secret value.

The TPM generates the sessionKey from the recovered secret value.

No authorization is required for tpmKey or bind.

NOTE 2 The justification for using tpmKey without providing authorization is that the result of using the key is not available to the caller, except indirectly through the sessionKey. This does not represent a point of attack on the value of the key. If the caller attempts to use the session without knowing the sessionKey value, it is an authorization failure that will trigger the dictionary attack logic.

The entity referenced with the bind parameter contributes an authorization value to the sessionKey generation process.

If both tpmKey and bind are TPM_ALG_NULL, then sessionKey is set to the Empty Buffer. If tpmKey is not TPM_ALG_NULL, then encryptedSalt is used in the computation of sessionKey. If bind is not TPM_ALG_NULL, the authValue of bind is used in the sessionKey computation.

If symmetric specifies a block cipher, then TPM_ALG_CFB is the only allowed value for the mode field in the symmetric parameter (TPM_RC_MODE).

This command starts an authorization session and returns the session handle along with an initial nonceTPM in the response.

If the TPM does not have a free slot for an authorization session, it shall return TPM_RC_SESSION_HANDLES.

If the TPM implements a “gap” scheme for assigning contextID values, then the TPM shall return TPM_RC_CONTEXT_GAP if creating the session would prevent recycling of old saved contexts (See “Context Management” in Part 1).

If tpmKey is not TPM_ALG_NULL then encryptedSalt shall be a TPM2B_ENCRYPTED_SECRET of the proper type for tpmKey. The TPM shall return TPM_RC_VALUE if:

a) tpmKey references an RSA key and

1) encryptedSalt does not contain a value that is the size of the public modulus of tpmKey,
2) encryptedSalt has a value that is greater than the public modulus of tpmKey,
3) encryptedSalt is not a properly encode OAEP value, or
4) the decrypted salt value is larger than the size of the digest produced by the nameAlg of tpmKey; or
b) tpmKey references an ECC key and encryptedSalt

1) does not contain a TPMS_ECC_POINT or
2) is not a point on the curve of tpmKey;

NOTE 3 When ECC is used, the point multiply process produces a value (Z) that is used in a KDF to produce the final secret value. The size of the secret value is an input parameter to the KDF and the result will be set to be the size of the digest produced by the nameAlg of tpmKey.

c) tpmKey references a symmetric block cipher or a keyedHash object and encryptedSalt contains a value that is larger than the size of the digest produced by the nameAlg of tpmKey.

For all session types, this command will cause initialization of the sessionKey and may establish binding between the session and an object (the bind object). If sessionType is TPM_SE_POLICY or TPM_SE_TRIAL, the additional session initialization is:

- set policySession→policyDigest to a Zero Digest (the digest size for policySession→policyDigest is the size of the digest produced by authHash);
- authorization may be given at any locality;
- authorization may apply to any command code;
- authorization may apply to any command parameters or handles;
- the authorization has no time limit;
- an authValue is not needed when the authorization is used;
- the session is not bound;
- the session is not an audit session; and
- the time at which the policy session was created is recorded.

Additionally, if sessionType is TPM_SE_TRIAL, the session will not be usable for authorization but can be used to compute the authPolicy for an object.

NOTE 4 Although this command changes the session allocation information in the TPM, it does not invalidate a saved context. That is, TPM2_Shutdown() is not required after this command in order to re-establish the orderly state of the TPM. This is because the created context will occupy an available slot in the TPM and sessions in the TPM do not survive any TPM2_Startup(). However, if a created session is context saved, the orderly state does change.

The TPM shall return TPM_RC_SIZE if nonceCaller is less than 16 octets or is greater than the size of the digest produced by authHash.
## 13.1.2 Command and Response

### Table 15 — TPM2_StartAuthSession Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_StartAuthSession</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>tpmKey</td>
<td>handle of a loaded decrypt key used to encrypt salt</td>
</tr>
<tr>
<td>TPMI_DH_ENTITY+</td>
<td>bind</td>
<td>entity providing the authValue</td>
</tr>
<tr>
<td>TPM2B_NONCE</td>
<td>nonceCaller</td>
<td>initial nonceCaller, sets nonce size for the session shall be at least 16 octets</td>
</tr>
<tr>
<td>TPM2B_ENCRYPTED_SECRET</td>
<td>encryptedSalt</td>
<td>value encrypted according to the type of tpmKey</td>
</tr>
<tr>
<td>TPM_SE</td>
<td>sessionType</td>
<td>indicates the type of the session; simple HMAC or policy (including a trial policy)</td>
</tr>
<tr>
<td>TPMT_SYM_DEF+</td>
<td>symmetric</td>
<td>the algorithm and key size for parameter encryption may select TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPMI_ALG_HASH</td>
<td>authHash</td>
<td>hash algorithm to use for the session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Shall be a hash algorithm supported by the TPM and not TPM_ALG_NULL</td>
</tr>
</tbody>
</table>

### Table 16 — TPM2_StartAuthSession Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMI_SH_AUTH_SESSION</td>
<td>sessionHandle</td>
<td>handle for the newly created session</td>
</tr>
<tr>
<td>TPM2B_NONCE</td>
<td>nonceTPM</td>
<td>the initial nonce from the TPM, used in the computation of the sessionKey</td>
</tr>
</tbody>
</table>
13.1.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "StartAuthSession_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td><code>tpmKey</code> does not reference a decrypt key</td>
</tr>
<tr>
<td>TPM_RC_CONTEXT_GAP</td>
<td>the difference between the most recently created active context and</td>
</tr>
<tr>
<td></td>
<td>the oldest active context is at the limits of the TPM</td>
</tr>
<tr>
<td>TPM_RC_HANDLE</td>
<td>input decrypt key handle only has public portion loaded</td>
</tr>
<tr>
<td>TPM_RC_MODE</td>
<td><code>symmetric</code> specifies a block cipher but the mode is not</td>
</tr>
<tr>
<td></td>
<td>TPM_ALG_CFB.</td>
</tr>
<tr>
<td>TPM_RC_SESSION_HANDLES</td>
<td>no session handle is available</td>
</tr>
<tr>
<td>TPM_RC_SESSION_MEMORY</td>
<td>no more slots for loading a session</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>nonce less than 16 octets or greater than the size of the digest</td>
</tr>
<tr>
<td></td>
<td>produced by <code>authHash</code></td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>secret size does not match decrypt key type; or the recovered secret</td>
</tr>
<tr>
<td></td>
<td>is larger than the digest size of the <code>nameAlg</code> of <code>tpmKey</code>; or, for an</td>
</tr>
<tr>
<td></td>
<td>RSA decrypt key, if <code>encryptedSecret</code> is greater than the public</td>
</tr>
<tr>
<td></td>
<td>exponent of <code>tpmKey</code>.</td>
</tr>
</tbody>
</table>
```

```c
TPM_RC

TPM2_StartAuthSession(
    StartAuthSession_In *in,     // IN: input parameter buffer
    StartAuthSession_Out *out    // OUT: output parameter buffer
)
{
    TPM_RC                   result = TPM_RC_SUCCESS;
    OBJECT                   *tpmKey;   // TPM key for decrypt salt
    SESSION                  *session;  // session internal data
    TPM2B_DATA               salt;

    // Input Validation

    // Check input nonce size. IT should be at least 16 bytes but not larger
    // than the digest size of session hash.
    if( in->nonceCaller.t.size < 16
        || in->nonceCaller.t.size > CryptGetHashDigestSize(in->authHash))
        return TPM_RC_SIZE + RC_StartAuthSession_nonceCaller;

    // If an decrypt key is passed in, check its validation
    if(in->tpmKey != TPM_RH_NULL)
    {
        // secret size cannot be 0
        if(in->encryptedSalt.t.size == 0)
            return TPM_RC_VALUE + RC_StartAuthSession_encryptedSalt;

        // Get pointer to loaded decrypt key
        tpmKey = ObjectGet(in->tpmKey);

        // Decrypting salt requires accessing the private portion of a key.
        // Therefore, tpmKey can not be a key with only public portion loaded
        if(tpmKey->attributes.publicOnly)
            return TPM_RC_HANDLE + RC_StartAuthSession_tpmKey;
```
// HMAC session input handle check.
// tpmKey should be a decryption key
if (tpmKey->publicArea.objectAttributes.decrypt != SET)
    return TPM_RC_ATTRIBUTES + RC_StartAuthSession_tpmKey;

// Secret Decryption. A TPM_RC_VALUE, TPM_RC_KEY or Unmarshal errors
// may be returned at this point
result = CryptSecretDecrypt(in->tpmKey, &in->nonceCaller, "SECRET",
                           &in->encryptedSalt, &salt);
if (result != TPM_RC_SUCCESS)
    return TPM_RC_VALUE + RC_StartAuthSession_encryptedSalt;

else
{
    // secret size must be 0
    if (in->encryptedSalt.t.size != 0)
        return TPM_RC_VALUE + RC_StartAuthSession_encryptedSalt;
    salt.t.size = 0;
}

// If 'symmetric' is a symmetric block cipher (not TPM_ALG_NULL or TPM_ALG_XOR)
// then the mode must be CFB.
if (    in->symmetric.algorithm != TPM_ALG_NULL
     && in->symmetric.algorithm != TPM_ALG_XOR
     && in->symmetric.mode.sym != TPM_ALG_CFB)
    return TPM_RC_MODE + RC_StartAuthSession_symmetric;

// Internal Data Update
// Create internal session structure. TPM_RC_CONTEXT_GAP, TPM_RC_NO_HANDLES
// or TPM_RC_SESSION_MEMORY errors may be returned at this point.
// The detailed actions for creating the session context are not shown here
// as the details are implementation dependent
// SessionCreate sets the output handle
result = SessionCreate(in->sessionType, in->authHash,
                       &in->nonceCaller, &in->symmetric,
                       in->bind, &salt, &out->sessionHandle);
if (result != TPM_RC_SUCCESS)
    return result;

// Command Output
// Get session pointer
session = SessionGet(out->sessionHandle);

// Copy nonceTPM
out->nonceTPM = session->nonceTPM;

return TPM_RC_SUCCESS;
13.2 TPM2_PolicyRestart

13.2.1 General Description

This command allows a policy authorization session to be returned to its initial state. This command is used after the TPM returns TPM_RC_PCR_CHANGED. That response code indicates that a policy will fail because the PCR have changed after TPM2_PolicyPCR() was executed. Restarting the session allows the authorizations to be replayed because the session restarts with the same nonceTPM. If the PCR are valid for the policy, the policy may then succeed.

This command does not reset the policy ID or the policy start time.
13.2.2 Command and Response

Table 17 — TPM2_PolicyRestart Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyRestart</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>sessionHandle</td>
<td>the handle for the policy session</td>
</tr>
</tbody>
</table>

Table 18 — TPM2_PolicyRestart Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
13.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyRestart_fp.h"

TPM_RC
TPM2_PolicyRestart(
    PolicyRestart_In *in  // IN: input parameter list
)
{
    SESSION *session;
    BOOL wasTrialSession;

    // Internal Data Update
    session = SessionGet(in->sessionHandle);
    wasTrialSession = session->attributes.isTrialPolicy == SET;

    // Initialize policy session
    SessionResetPolicyData(session);
    session->attributes.isTrialPolicy = wasTrialSession;

    return TPM_RC_SUCCESS;
}
```
14 Object Commands

14.1 TPM2_Create

14.1.1 General Description

This command is used to create an object that can be loaded into a TPM using TPM2_Load(). If the command completes successfully, the TPM will create the new object and return the object's creation data (creationData), its public area (outPublic), and its encrypted sensitive area (outPrivate). Preservation of the returned data is the responsibility of the caller. The object will need to be loaded (TPM2_Load()) before it may be used.

TPM2B_PUBLIC template (inPublic) contains all of the fields necessary to define the properties of the new object. The setting for these fields is defined in “Public Area Template” in Part 1 and “TPMA_OBJECT” in Part 2.

The parentHandle parameter shall reference a loaded decryption key that has both the public and sensitive area loaded.

When defining the object, the caller provides a template structure for the object in a TPM2B_PUBLIC structure (inPublic), an initial value for the object’s authValue (inSensitive.authValue), and, if the object is a symmetric object, an optional initial data value (inSensitive.data). The TPM shall validate the consistency of inPublic.attributes according to the Creation rules in “TPMA_OBJECT” in Part 2.

The sensitive parameter may be encrypted using parameter encryption.

The methods in this clause are used by both TPM2_Create() and TPM2_CreatePrimary(). When a value is indicated as being TPM-generated, the value is filled in by bits from the RNG if the command is TPM2_Create() and with values from KDFa() if the command is TPM2_CreatePrimary(). The parameters of each creation value are specified in Part 1.

The sensitiveDataOrigin attribute of inPublic shall be SET if inSensitive.data is an Empty Buffer and CLEAR if inSensitive.data is not an Empty Buffer or the TPM shall return TPM_RC_ATTRIBUTES.

The TPM will create new data for the sensitive area and compute a TPMT_PUBLIC.unique from the sensitive area based on the object type:

a) For a symmetric key:

1) If inSensitive.data is the Empty Buffer, a TPM-generated key value is placed in the new object's TPMT_SENSITIVE.sensitive.sym. The size of the key will be determined by inPublic.publicArea.parameters.

2) If inSensitive.data is not the Empty Buffer, the TPM will validate that the size of inSensitive.data is no larger than the key size indicated in the inPublic template (TPM_RC_SIZE) and copy the inSensitive.data to TPMT_SENSITIVE.sensitive.sym of the new object.

3) A TPM-generated obfuscation value is placed in TPMT_SENSITIVE.sensitive.seedValue. The size of the obfuscation value is the size of the digest produced by the nameAlg in inPublic. This value prevents the public unique value from leaking information about the sensitive area.

4) The TPMT_PUBLIC.unique.sym.buffer value for the new object is then generated, as shown in equation (1) below, by hashing the key and obfuscation values in the TPMT_SENSITIVE with the nameAlg of the object.

\[ unique := H_{nameAlg}(sensitive.seedValue.buffer \mid| sensitive.any.buffer) \]  (1)

b) If the Object is an asymmetric key:

1) If sensitive.data is not the Empty Buffer, then the TPM shall return TPM_RC_VALUE.
2) A TPM-generated private key value is created with the size determined by the parameters of inPublic.publicArea.parameters.

3) If the key is a Storage Key, a TPM-generated TPMT_SENSITIVE.symKey value is created; otherwise, TPMT_SENSITIVE.symKey.size is set to zero.

4) The public unique value is computed from the private key according to the methods of the key type.

5) If the key is an ECC key and the scheme required by the curveID is not the same as scheme in the public area of the template, then the TPM shall return TPM_RC_SCHEME.

6) If the key is an ECC key and the KDF required by the curveID is not the same as kdf in the public area of the template, then the TPM shall return TPM_RC_KDF.

NOTE 1 There is currently no command in which the caller may specify the KDF to be used with an ECC decryption key. Since there is no use for this capability, the reference implementation requires that the kdf in the template be set to TPM_ALG_NULL or TPM_RC_KDF is returned.

c) If the Object is a keyedHash object:

1) If inSensitive.data is an Empty Buffer, and neither sign nor decrypt is SET in inPublic.attributes, the TPM shall return TPM_RC_ATTRIBUTES. This would be a data object with no data.

2) If inSensitive.data is not an Empty Buffer, the TPM will copy the inSensitive.data to TPMT_SENSITIVE.sensitive of the new object.

NOTE 2 The size of inSensitive.data is limited to be no larger than the largest value of TPMT_SENSITIVE.sensitive.bits.data by MAX_SYM_DATA.

3) If inSensitive.data is an Empty Buffer, a TPM-generated key value that is the size of the digest produced by the nameAlg in inPublic is placed in TPMT_SENSITIVE.sensitive.any.buffer.

4) A TPM-generated obfuscation value that is the size of the digest produced by the nameAlg of inPublic is placed in TPMT_SENSITIVE.symKey.buffer.

5) The TPMT_PUBLIC.unique.sym.buffer value for the new object is then generated, as shown in equation (1) above, by hashing the key and obfuscation values in the TPMT_SENSITIVE with the nameAlg of the object.

For TPM2_Load(), the TPM will apply normal symmetric protections to the created TPMT_SENSITIVE to create outPublic.

NOTE 3 The encryption key is derived from the symmetric seed in the sensitive area of the parent.

In addition to outPublic and outPrivate, the TPM will build a TPMS_CREATION_DATA structure for the object. TPMS_CREATION_DATA.outsideInfo is set to outsideInfo. This structure is returned in creationData. Additionally, the digest of this structure is returned in creationHash, and, finally, a TPMT_TK_CREATION is created so that the association between the creation data and the object may be validated by TPM2_CertifyCreation().

If the object being created is a Storage Key and inPublic.objectAttributes.fixedParent is SET, then the algorithms of inPublic are required to match those of the parent. The algorithms that must match are inPublic.type, inPublic.nameAlg, and inPublic.parameters. If inPublic.type does not match, the TPM shall return TPM_RC_TYPE. If inPublic.nameAlg does not match, the TPM shall return TPM_RC_HASH. If inPublic.parameters does not match, the TPM shall return TPM_RC_ASSYMETRIC. The TPM shall not differentiate between mismatches of the components of inPublic.parameters.

EXAMPLE If the inPublic.parameters.ecc.symmetric.algorithm does not match the parent, the TPM shall return TPM_RCASYMMETRIC rather than TPM_RC_SYMMETRIC.
14.1.2 Command and Response

### Table 19 — TPM2_Create Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Create</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@parentHandle</td>
<td>handle of parent for new object</td>
</tr>
<tr>
<td></td>
<td>Auth Index: 1</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td>Auth Role: USER</td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_SENSITIVE_CREATE</td>
<td>inSensitive</td>
<td>the sensitive data</td>
</tr>
<tr>
<td>TPM2B_PUBLIC</td>
<td>inPublic</td>
<td>the public template</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>outsideInfo</td>
<td>data that will be included in the creation data for this object to provide permanent, verifiable linkage between this object and some object owner data</td>
</tr>
<tr>
<td>TPML_PCR_SELECTION</td>
<td>creationPCR</td>
<td>PCR that will be used in creation data</td>
</tr>
</tbody>
</table>

### Table 20 — TPM2_Create Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_PRIVATE</td>
<td>outPrivate</td>
<td>the private portion of the object</td>
</tr>
<tr>
<td>TPM2B_PUBLIC</td>
<td>outPublic</td>
<td>the public portion of the created object</td>
</tr>
<tr>
<td>TPM2B_CREATION_DATA</td>
<td>creationData</td>
<td>contains a TPMS_CREATION_DATA</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>creationHash</td>
<td>digest of creationData using nameAlg of outPublic</td>
</tr>
<tr>
<td>TPMT_TK_CREATION</td>
<td>creationTicket</td>
<td>ticket used by TPM2_CertifyCreation() to validate that the creation data was produced by the TPM</td>
</tr>
</tbody>
</table>
14.1.3 Detailed Actions

1 #include "InternalRoutines.h"
2 #include "Object_spt_fp.h"
3 #include "Create_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC ASYMMETRIC</td>
<td>non-duplicable storage key and its parent have different public params</td>
</tr>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>sensitiveDataOrigin is CLEAR when 'sensitive. data' is an Empty Buffer, or is SET when 'sensitive. data' is not empty; 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>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_RC_KEY_SIZE</td>
<td>key size in public area for symmetric key differs from the size in the sensitive creation area; may also be returned if the TPM does not allow the key size to be used for a Storage Key</td>
</tr>
<tr>
<td>TPM_RC_RANGE</td>
<td>FOr() an RSA key, the exponent value is not supported.</td>
</tr>
<tr>
<td>TPM_RC_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_SIZE</td>
<td>size of public auth policy or sensitive auth value does not match digest size of the name algorithm sensitive data size for the keyed hash object is larger than is allowed for the scheme</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 TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>unknown object type; non-duplicable storage key and its parent have different types; parentHandle does not reference a restricted decryption key in the storage hierarchy with both public and sensitive portion loaded</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>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>there is no free slot for the object. This implementation does not return this error.</td>
</tr>
</tbody>
</table>

```
TPM_RC_CREATE
TPM2_Create(
    Create_In   *in,       // IN: input parameter list
    Create_Out  *out       // OUT: output parameter list
)
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE          sensitive;
    TPM2B_NAME              name;
```
14 // Input Validation
15 
16 OBJECT *parentObject;
17 
18 parentObject = ObjectGet(in->parentHandle);
19 
20 // Does parent have the proper attributes?
21 if(!AreAttributesForParent(parentObject))
22     return TPM_RC_TYPE + RC_Create_parentHandle;
23 
24 // The sensitiveDataOrigin attribute must be consistent with the setting of
25 // the size of the data object in inSensitive.
26 if( (in->inPublic.t.publicArea.objectAttributes.sensitiveDataOrigin == SET)
27         != (in->inSensitive.t.sensitive.data.t.size == 0))
28     // Mismatch between the object attributes and the parameter.
29     return TPM_RC_ATTRIBUTES + RC_Create_inSensitive;
30 
31 // Check attributes in input public area. TPM_RC_ASMETRIC, TPM_RC_ATTRIBUTES,
32 // TPM_RC_HASH, TPM_RC_KDF, TPM_RC_SCHEMA, TPM_RC_SIZE, TPM_RC_SYMMENTRIC,
33 // or TPM_RC_TYPE error may be returned at this point.
34 result = PublicAttributesValidation(FALSE, in->parentHandle,
35                                          &in->inPublic.t.publicArea);
36 if(result != TPM_RC_SUCCESS)
37     return RcSafeAddToResult(result, RC_Create_inPublic);
38 
39 // Validate the sensitive area values
40 if( MemoryRemoveTrailingZeros(&in->inSensitive.t.sensitive.userAuth)
41         > CryptGetHashDigestSize(in->inPublic.t.publicArea.nameAlg))
42     return TPM_RC_SIZE + RC_Create_inSensitive;
43 
44 // Command Output
45 
46 // Create object crypto data
47 result = CryptCreateObject(in->parentHandle, &in->inPublic.t.publicArea,
48                              &in->inSensitive.t.sensitive, &sensitive);
49 if(result != TPM_RC_SUCCESS)
50     return result;
51 
52 // Fill in creation data
53 FillInCreationData(in->parentHandle, in->inPublic.t.publicArea.nameAlg,
54                     &in->creationPCR, &in->outsideInfo,
55                     &out->creationData, &out->creationHash);
56 
57 // Copy public area from input to output
58 out->outPublic.t.publicArea = in->inPublic.t.publicArea;
59 
60 // Compute name from public area
61 ObjectComputeName(&(out->outPublic.t.publicArea), &name);
62 
63 // Compute creation ticket
64 TicketComputeCreation(EntityGetHierarchy(in->parentHandle), &name,
65                        &out->creationHash, &out->creationTicket);
66 
67 // Prepare output private data from sensitive
68 SensitiveToPrivate(&sensitive, &name, in->parentHandle,
69                        out->outPublic.t.publicArea.nameAlg,
70                        &out->outPrivate);
71 
72 return TPM_RC_SUCCESS;
14.2 TPM2_Load

14.2.1 General Description

This command is used to load objects into the TPM. This command is used when both a TPM2B_PUBLIC and TPM2B_PRIVATE are to be loaded. If only a TPM2B_PUBLIC is to be loaded, the TPM2_LoadExternal command is used.

NOTE 1 Loading an object is not the same as restoring a saved object context.

The object’s TPMA_OBJECT attributes will be checked according to the rules defined in “TPMA_OBJECT” in Part 2 of this specification.

Objects loaded using this command will have a Name. The Name is the concatenation of nameAlg and the digest of the public area using the nameAlg.

NOTE 2 nameAlg is a parameter in the public area of the inPublic structure.

If inPrivate.size is zero, the load will fail.

After inPrivate.buffer is decrypted using the symmetric key of the parent, the integrity value shall be checked before the sensitive area is used, or unmarshaled.

NOTE 3 Checking the integrity before the data is used prevents attacks on the sensitive area by fuzzing the data and looking at the differences in the response codes.

The command returns a handle for the loaded object and the Name that the TPM computed for inPublic.public (that is, the digest of the TPMT_PUBLIC structure in inPublic).

NOTE 4 The TPM-computed Name is provided as a convenience to the caller for those cases where the caller does not implement the hash algorithms specified in the nameAlg of the object.

NOTE 5 The returned handle is associated with the object until the object is flushed (TPM2_FlushContext) or until the next TPM2_Startup.

For all objects, the size of the key in the sensitive area shall be consistent with the key size indicated in the public area or the TPM shall return TPM_RC_KEY_SIZE.

Before use, a loaded object shall be checked to validate that the public and sensitive portions are properly linked, cryptographically. Use of an object includes use in any policy command. If the parts of the object are not properly linked, the TPM shall return TPM_RC_BINDING.

EXAMPLE 1 For a symmetric object, the unique value in the public area shall be the digest of the sensitive key and the obfuscation value.

EXAMPLE 2 For a two-prime RSA key, the remainder when dividing the public modulus by the private key shall be zero and it shall be possible to form a private exponent from the two prime factors of the public modulus.

EXAMPLE 3 For an ECC key, the public point shall be f(x) where x is the private key.
### 14.2.2 Command and Response

#### Table 21 — TPM2_Load Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Load</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@parentHandle</td>
<td>TPM handle of parent key; shall not be a reserved handle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_PRIVATE</td>
<td>inPrivate</td>
<td>the private portion of the object</td>
</tr>
<tr>
<td>TPM2B_PUBLIC</td>
<td>inPublic</td>
<td>the public portion of the object</td>
</tr>
</tbody>
</table>

#### Table 22 — TPM2_Load Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM_HANDLE</td>
<td>objectHandle</td>
<td>handle for the loaded object</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>name</td>
<td>Name of the loaded object</td>
</tr>
</tbody>
</table>
14.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "Load_fp.h"
#include "Object_spt_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RCASYMMETRIC</td>
<td>storage key with different asymmetric type than parent</td>
</tr>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>inPublic attributes are not allowed with selected parent</td>
</tr>
<tr>
<td>TPM_RC_BINDING</td>
<td>inPrivate and inPublic are not cryptographically bound</td>
</tr>
<tr>
<td>TPM_RC_HASH</td>
<td>incorrect hash selection for signing key</td>
</tr>
<tr>
<td>TPM_RC_INTEGRITY</td>
<td>HMAC on inPrivate was not valid</td>
</tr>
<tr>
<td>TPM_RC_KDF</td>
<td>KDF selection not allowed</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>the size of the object's unique field is not consistent with the indicated size in the object’s parameters</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>no available object slot</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>the signing scheme is not valid for the key</td>
</tr>
<tr>
<td>TPM_RC_SENSITIVE</td>
<td>the inPrivate did not unmarshal correctly</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>inPrivate missing, or authPolicy size for inPublic or is not valid</td>
</tr>
<tr>
<td>TPM_RC_SYMMETRIC</td>
<td>symmetric algorithm not provided when required</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>parentHandle is not a storage key, or the object to load is a storage key but its parameters do not match the parameters of the parent.</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>decryption failure</td>
</tr>
</tbody>
</table>

```
TPM_RC
TPM2_Load(
    Load_In *in,          // IN: input parameter list
    Load_Out *out)        // OUT: output parameter list
{
    TPM_RC                   result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE           sensitive;
    TPMIHierarchy            hierarchy;
    OBJECT                    *parentObject = NULL;
    BOOL                      skipChecks = FALSE;

    // Input Validation
    if(in->inPrivate.t.size == 0)
        return TPM_RC_SIZE + RC_Load_inPrivate;
    parentObject = ObjectGet(in->parentHandle);
    // Is the object that is being used as the parent actually a parent.
    if(!AreAttributesForParent(parentObject))
        return TPM_RC_TYPE + RC_Load_parentHandle;
    // If the parent is fixedTPM, then the attributes of the object
    // are either "correct by construction" or were validated
    // when the object was imported. If they pass the integrity
    // check, then the values are valid
    if(parentObject->publicArea.objectAttributes.fixedTPM)
        skipChecks = TRUE;
```
```c
else {
    // If parent doesn't have fixedTPM SET, then this can't have
    // fixedTPM SET.
    if (in->inPublic.t.publicArea.objectAttributes.fixedTPM == SET) {
        return TPM_RC_ATTRIBUTES + RC_Load_inPublic;
    }
    // Perform self check on input public area. A TPM_RC_SIZE, TPM_RC_SCHEME,
    // TPM_RC_VALUE, TPM_RC_SYMMETRIC, TPM_RC_TYPE, TPM_RC_HASH,
    // TPM_RC_ASYMMETRIC, TPM_RC_ATTRIBUTES or TPM_RC_KDF error may be returned
    // at this point
    result = PublicAttributesValidation(TRUE, in->parentHandle, &in->inPublic.t.publicArea);
    if (result != TPM_RC_SUCCESS) {
        return RcSafeAddToResult(result, RC_Load_inPublic);
    }

    // Compute the name of object
    ObjectComputeName(&in->inPublic.t.publicArea, &out->name);

    // Retrieve sensitive data. PrivateToSensitive() may return TPM_RC_INTEGRITY or
    // TPM_RC_SENSITIVE
    // errors may be returned at this point
    result = PrivateToSensitive(&in->inPrivate, &out->name, in->parentHandle, in->inPublic.t.publicArea.nameAlg, &sensitive);
    if (result != TPM_RC_SUCCESS) {
        return RcSafeAddToResult(result, RC_Load_inPrivate);
    }

    // Internal Data Update
    // Get hierarchy of parent
    hierarchy = ObjectGetHierarchy(in->parentHandle);

    // Create internal object. A lot of different errors may be returned by this
    // loading operation as it will do several validations, including the public
    // binding check
    result = ObjectLoad(hierarchy, &in->inPublic.t.publicArea, &sensitive, &out->name, in->parentHandle, skipChecks, &out->objectHandle);
    if (result != TPM_RC_SUCCESS) {
        return result;
    }

    return TPM_RC_SUCCESS;
}
```
14.3 TPM2_LoadExternal

14.3.1 General Description

This command is used to load an object that is not a Protected Object into the TPM. The command allows loading of a public area or both a public and sensitive area.

NOTE 1
Typical use for loading a public area is to allow the TPM to validate an asymmetric signature. Typical use for loading both a public and sensitive area is to allow the TPM to be used as a crypto accelerator.

Load of a public external object area allows the object be associated with a hierarchy so that the correct algorithms may be used when creating tickets. The hierarchy parameter provides this association. If the public and sensitive portions of the object are loaded, hierarchy is required to be TPM_RH_NULL.

NOTE 2
If both the public and private portions of an object are loaded, the object is not allowed to appear to be part of a hierarchy.

The object's TPMA_OBJECT attributes will be checked according to the rules defined in "TPMA_OBJECT" in Part 2. In particular, fixedTPM, fixedParent, and restricted shall be CLEAR if inPrivate is not the Empty Buffer.

NOTE 3
The duplication status of a public key needs to be able to be the same as the full key which may be resident on a different TPM. If both the public and private parts of the key are loaded, then it is not possible for the key to be either fixedTPM or fixedParent, since, its private area would not be available in the clear to load.

Objects loaded using this command will have a Name. The Name is the nameAlg of the object concatenated with the digest of the public area using the nameAlg. The Qualified Name for the object will be the same as its Name. The TPM will validate that the authPolicy is either the size of the digest produced by nameAlg or the Empty Buffer.

NOTE 4
If nameAlg is TPM_ALG_NULL, then the Name is the Empty Buffer. When the authorization value for an object with no Name is computed, no Name value is included in the HMAC. To ensure that these unnamed entities are not substituted, they should have an authValue that is statistically unique.

NOTE 5
The digest size for TPM_ALG_NULL is zero.

If the nameAlg is TPM_ALG_NULL, the TPM shall not verify the cryptographic binding between the public and sensitive areas, but the TPM will validate that the size of the key in the sensitive area is consistent with the size indicated in the public area. If it is not, the TPM shall return TPM_RC_KEY_SIZE.

NOTE 6
For an ECC object, the TPM will verify that the public key is on the curve of the key before the public area is used.

If nameAlg is not TPM_ALG_NULL, then the same consistency checks between inPublic and inPrivate are made as for TPM2_Load().

NOTE 7
Consistency checks are necessary because an object with a Name needs to have the public and sensitive portions cryptographically bound so that an attacker cannot mix public and sensitive areas.

The command returns a handle for the loaded object and the Name that the TPM computed for inPublic.public (that is, the TPMT_PUBLIC structure in inPublic).

NOTE 8
The TPM-computed Name is provided as a convenience to the caller for those cases where the caller does not implement the hash algorithm specified in the nameAlg of the object.
The *hierarchy* parameter associates the external object with a hierarchy. External objects are flushed when their associated hierarchy is disabled. If *hierarchy* is `TPM_RH_NULL`, the object is part of no hierarchy, and there is no implicit flush.

If *hierarchy* is `TPM_RH_NULL` or *nameAlg* is `TPM_ALG_NULL`, a ticket produced using the object shall be a NULL Ticket.

**EXAMPLE**

If a key is loaded with hierarchy set to `TPM_RH_NULL`, then `TPM2_VerifySignature()` will produce a NULL Ticket of the required type.

External objects are Temporary Objects. The saved external object contexts shall be invalidated at the next TPM Reset.
14.3.2 Command and Response

Table 23 — TPM2_LoadExternal Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_LoadExternal</td>
</tr>
<tr>
<td>TPM2B_SENSITIVE</td>
<td>inPrivate</td>
<td>the sensitive portion of the object (optional)</td>
</tr>
<tr>
<td>TPM2B_PUBLIC+</td>
<td>inPublic</td>
<td>the public portion of the object</td>
</tr>
<tr>
<td>TPMI_RH_HIERARCHY+</td>
<td>hierarchy</td>
<td>hierarchy with which the object area is associated</td>
</tr>
</tbody>
</table>

Table 24 — TPM2_LoadExternal Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM_HANDLE</td>
<td>objectHandle</td>
<td>handle for the loaded object</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>name</td>
<td>name of the loaded object</td>
</tr>
</tbody>
</table>
14.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "LoadExternal_fp.h"
#include "Object_spt_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>'fixedParent' and fixedTPM must be CLEAR on an on an external key if both public and sensitive portions are loaded</td>
</tr>
<tr>
<td>TPM_RC_BINDING</td>
<td>the inPublic and inPrivate structures are not cryptographically bound.</td>
</tr>
<tr>
<td>TPM_RC_HASH</td>
<td>incorrect hash selection for signing key</td>
</tr>
<tr>
<td>TPM_RC_HIERARCHY</td>
<td>hierarchy is turned off, or only NULL hierarchy is allowed when loading public and private parts of an object</td>
</tr>
<tr>
<td>TPM_RC_KDF</td>
<td>incorrect KDF selection for decrypting keyedHash object</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>the size of the object's unique field is not consistent with the indicated size in the object's parameters</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>if there is no free slot for an object</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>the signing scheme is not valid for the key</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>authPolicy is not zero and is not the size of a digest produced by the object's nameAlg TPM_RH_NULL hierarchy</td>
</tr>
<tr>
<td>TPM_RC_SYMMETRIC</td>
<td>symmetric algorithm not provided when required</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>inPublic and inPrivate are not the same type</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_LoadExternal(
  LoadExternal_In     *in,    // IN: input parameter list
  LoadExternal_Out    *out    // OUT: output parameter list
){
  TPM_RC               result;
  TPMFT_SENSITIVE     *sensitive;
  BOOL                 skipChecks;
  // Input Validation
  if(!HierarchyIsEnabled(in->hierarchy))
    return TPM_RC_HIERARCHY + RC_LoadExternal_hierarchy;
  if(in->inPublic.t.publicArea.authPolicy.t.size != 0
     && in->inPublic.t.publicArea.authPolicy.t.size !=
     CryptGetHashDigestSize(in->inPublic.t.publicArea.nameAlg))
    return TPM_RC_SIZE + RC_LoadExternal_inPublic;
  // For loading an object with both public and sensitive
  if(in->inPrivate.t.size != 0)
    { // An external object can only be loaded at TPM_RH_NULL hierarchy
      if(in->hierarchy != TPM_RH_NULL)
        return TPM_RC_HIERARCHY + RC_LoadExternal_hierarchy;
      // An external object with a sensitive area must have fixedTPM == CLEAR
      // fixedParent == CLEAR, and must have restrict CLEAR so that it does not
```
// appear to be a key that was created by this TPM.
if (in->inPublic.t.publicArea.objectAttributes.fixedTPM != CLEAR
    || in->inPublic.t.publicArea.objectAttributes.fixedParent != CLEAR
    || in->inPublic.t.publicArea.objectAttributes.restricted != CLEAR)
    return TPM_RC_ATTRIBUTES + RC_LoadExternal_inPublic;
}

// Validate the scheme parameters
result = SchemeChecks(TRUE, TPM_RH_NULL, &in->inPublic.t.publicArea);
if (result != TPM_RC_SUCCESS)
    return RcSafeAddToResult(result, RC_LoadExternal_inPublic);

// Internal Data Update
// Need the name to compute the qualified name
ObjectComputeName(&in->inPublic.t.publicArea, &out->name);
skipChecks = (in->inPublic.t.publicArea.nameAlg == TPM_ALG_NULL);

// If a sensitive area was provided, load it
if (in->inPrivate.t.size != 0)
sensitive = &in->inPrivate.t.sensitiveArea;
else
    sensitive = NULL;

// Create external object. A TPM_RC_BINDING, TPM_RC_KEY, TPM_RC_OBJECT_MEMORY
// or TPM_RC_TYPE error may be returned by ObjectLoad()
result = ObjectLoad(in->hierarchy, &in->inPublic.t.publicArea,
    sensitive, &out->name, TPM_RH_NULL, skipChecks,
    &out->objectHandle);
return result;
14.4 TPM2_ReadPublic

14.4.1 General Description

This command allows access to the public area of a loaded object. Use of the objectHandle does not require authorization.

NOTE Since the caller is not likely to know the public area of the object associated with objectHandle, it would not be possible to include the Name associated with objectHandle in the cpHash computation.

If objectHandle references a sequence object, the TPM shall return TPM_RC_SEQUENCE.
14.4.2 Command and Response

### Table 25 — TPM2_ReadPublic Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ReadPublic</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>objectHandle</td>
<td>TPM handle of an object Auth Index: None</td>
</tr>
</tbody>
</table>

### Table 26 — TPM2_ReadPublic Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_PUBLIC</td>
<td>outPublic</td>
<td>structure containing the public area of an object</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>name</td>
<td>name of the object</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>qualifiedName</td>
<td>the Qualified Name of the object</td>
</tr>
</tbody>
</table>
14.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ReadPublic_fp.h"

Error Returns | Meaning
-------------|------------------
TPM_RC_SEQUENCE | can not read the public area of a sequence object

TPM_RC
TPM2_ReadPublic(
  ReadPublic_In *in,       // IN: input parameter list
  ReadPublic_Out *out      // OUT: output parameter list
)
{
  OBJECT *object;

  // Input Validation
  object = ObjectGet(in->objectHandle);

  // Can not read public area of a sequence object
  if(ObjectIsSequence(object))
    return TPM_RC_SEQUENCE;

  // Command Output
  out->outPublic.t.size = TPMT_PUBLIC_Marshal(&object->publicArea, NULL, NULL);
  out->outPublic.t.publicArea = object->publicArea;
  out->name.t.size = ObjectGetName(in->objectHandle, &out->name.t.name);
  ObjectGetQualifiedName(in->objectHandle, &out->qualifiedName);

  return TPM_RC_SUCCESS;
}
```
14.5 TPM2_ActivateCredential

14.5.1 General Description

This command enables the association of a credential with an object in a way that ensures that the TPM has validated the parameters of the credentialed object.

If both the public and private portions of activateHandle and keyHandle are not loaded, then the TPM shall return TPM_RC_AUTH_UNAVAILABLE.

If keyHandle is not a Storage Key, then the TPM shall return TPM_RC_TYPE.

Authorization for activateHandle requires the ADMIN role.

The key associated with keyHandle is used to recover a seed from secret, which is the encrypted seed. The Name of the object associated with activateHandle and the recovered seed are used in a KDF to recover the symmetric key. The recovered seed (but not the Name) is used in a KDF to recover the HMAC key.

The HMAC is used to validate that the credentialBlob is associated with activateHandle and that the data in credentialBlob has not been modified. The linkage to the object associated with activateHandle is achieved by including the Name in the HMAC calculation.

If the integrity checks succeed, credentialBlob is decrypted and returned as certInfo.
## 14.5.2 Command and Response

### Table 27 — TPM2_ActivateCredential Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ActivateCredential handle of the object associated with certificate in credentialBlob Auth Index: 1 Auth Role: ADMIN</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@activateHandle</td>
<td>the data should be no larger than the size of the digest of the nameAlg associated with keyHandle</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@keyHandle</td>
<td>loaded key used to decrypt the TPMS_SENSITIVE in credentialBlob Auth Index: 2 Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_ID_OBJECT</td>
<td>credentialBlob</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ENCRYPTED_SECRET</td>
<td>secret</td>
<td>keyHandle algorithm-dependent encrypted seed that protects credentialBlob</td>
</tr>
</tbody>
</table>

### Table 28 — TPM2_ActivateCredential Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>certInfo</td>
<td>the decrypted certificate information the data should be no larger than the size of the digest of the nameAlg associated with keyHandle</td>
</tr>
</tbody>
</table>
14.5.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "ActivateCredential_fp.h"
#include "Object_spt_fp.h"

<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 decryption key</td>
</tr>
<tr>
<td>TPM_RC_ECC_POINT</td>
<td>secret is invalid (when keyHandle is an ECC key)</td>
</tr>
<tr>
<td>TPM_RC_INSUFFICIENT</td>
<td>secret is invalid (when keyHandle is an ECC key)</td>
</tr>
<tr>
<td>TPM_RC_INTEGRITY</td>
<td>credentialBlob fails integrity test</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>secret is invalid (when keyHandle is an ECC key)</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>secret size is invalid or the credentialBlob does not unmarshal correctly</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>keyHandle does not reference an asymmetric key.</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>secret is invalid (when keyHandle is an RSA key)</td>
</tr>
</tbody>
</table>

TPM2_ActivateCredential(
    TPM2_ActivateCredential_In *in,  // IN: input parameter list
    TPM2_ActivateCredential_Out *out  // OUT: output parameter list
) {
    TPM_RC result = TPM_RC_SUCCESS;
    OBJECT *object;  // decrypt key
    OBJECT *activateObject; // key associated with credential
    TPM2B_DATA data;  // credential data

    // Input Validation
    object = ObjectGet(in->keyHandle);
    activateObject = ObjectGet(in->activateHandle);

    // input decrypt key must be an asymmetric, restricted decryption key
    if (!CryptIsAsymAlgorithm(object->publicArea.type)
        || object->publicArea.objectAttributes.decrypt == CLEAR
        || object->publicArea.objectAttributes.restricted == CLEAR)
        return TPM_RC_TYPE + RC_ActivateCredential_keyHandle;

    // Decrypt input credential data via asymmetric decryption. A
    // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
    // point
    result = CryptSecretDecrypt(in->keyHandle, NULL,
                                "IDENTITY", &in->secret, &data);

    if(result != TPM_RC_SUCCESS)
        return TPM_RC_FAILURE;
```
return RcSafeAddToResult(result, RC_ActivateCredential_secret);  
}

// Retrieve secret data. A TPM_RC_INTEGRITY error or unmarshal errors may be returned at this point
result = CredentialToSecret(&in->credentialBlob,
                         &activateObject->name,
                         (TPM2B_SEED *) &data,
                         in->keyHandle,
                         &out->certInfo);
if (result != TPM_RC_SUCCESS)
    return RcSafeAddToResult(result,RC_ActivateCredential_credentialBlob);
return TPM_RC_SUCCESS;
14.6 TPM2_MakeCredential

14.6.1 General Description

This command allows the TPM to perform the actions required of a Certificate Authority (CA) in creating a
TPM2B_ID_OBJECT containing an activation credential.
The TPM will produce a TPM_ID_OBJECT according to the methods in “Credential Protection” in Part 1.
The loaded public area referenced by handle is required to be the public area of a Storage key, otherwise, the credential cannot be properly sealed.
This command does not use any TPM secrets nor does it require authorization. It is a convenience function, using the TPM to perform cryptographic calculations that could be done externally.
### 14.6.2 Command and Response

#### Table 29 — TPM2_MakeCredential Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>size</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_MakeCredential</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>handle</td>
<td>loaded public area, used to encrypt the sensitive area containing the credential key Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>credential</td>
<td>the credential information</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>objectName</td>
<td>Name of the object to which the credential applies</td>
</tr>
</tbody>
</table>

#### Table 30 — TPM2_MakeCredential Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ID_OBJECT</td>
<td>credentialBlob</td>
<td>the credential</td>
</tr>
<tr>
<td>TPM2B_ENCRYPTED_SECRET</td>
<td>secret</td>
<td><em>handle</em> algorithm-dependent data that wraps the key that encrypts credentialBlob</td>
</tr>
</tbody>
</table>
14.6.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "MakeCredential_fp.h"
#include "Object_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>handle referenced an ECC key that has a unique field that is not a point on the curve of the key</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>credential is larger than the digest size of Name algorithm of handle</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>handle does not reference an asymmetric decryption key</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_MakeCredential(
    MakeCredential_In       *in,       // IN: input parameter list
    MakeCredential_Out      *out       // OUT: output parameter list
)
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    OBJECT                  *object;
    TPM2B_DATA              data;

    // Input Validation
    object = ObjectGet(in->handle);
    if (CryptIsAsymAlgorithm(object->publicArea.type)
        || object->publicArea.objectAttributes.decrypt == CLEAR
        || object->publicArea.objectAttributes.restricted == CLEAR
    )
        return TPM_RC_TYPE + RC_MakeCredential_handle;
    // The credential information may not be larger than the digest size used for
    // the Name of the key associated with handle.
    if (in->credential.t.size > CryptGetHashDigestSize(object->publicArea.nameAlg))
        return TPM_RC_SIZE + RC_MakeCredential_credential;

    // Command Output
    out->secret.t.size = sizeof(out->secret.t.secret);
    result = CryptSecretEncrypt(in->handle, "IDENTITY", &data, &out->secret);
    if (result != TPM_RC_SUCCESS)
        return result;
    SecretToCredential(&in->credential, &in->objectName, (TPM2B_SEED *) &data,
                        in->handle, &out->credentialBlob);
    return TPM_RC_SUCCESS;
}
```
14.7 TPM2_Unseal

14.7.1 General Description

This command returns the data in a loaded Sealed Data Object.

NOTE A random, TPM-generated, Sealed Data Object may be created by the TPM with TPM2_Create() or TPM2_CreatePrimary() using the template for a Sealed Data Object. A Sealed Data Object is more likely to be created externally and imported (TPM2_IMPORT()) so that the data is not created by the TPM.

The returned value may be encrypted using authorization session encryption.

If either restricted, decrypt, or sign is SET in the attributes of itemHandle, then the TPM shall return TPM_RC_ATTRIBUTES. If the type of itemHandle is not TPM_ALG_KEYEDHASH, then the TPM shall return TPM_RC_TYPE.
14.7.2 Command and Response

Table 31 — TPM2_Unseal Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>Tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Unseal</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@itemHandle</td>
<td>handle of a loaded data object</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
</tbody>
</table>

Table 32 — TPM2_Unseal Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPMI_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>outData</td>
<td>unsealed data</td>
</tr>
</tbody>
</table>
|                 |                   | Size of *outData* is limited to be no more than 128 octets.
14.7.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Unseal_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td><code>itemHandle</code> has wrong attributes</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td><code>itemHandle</code> is not a KEYEDHASH data object</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_Unseal(Unseal_In *in, Unseal_Out *out)
{
    OBJECT *object;

    // Input Validation
    // Get pointer to loaded object
    object = ObjectGet(in->itemHandle);

    // Input handle must be a data object
    if(object->publicArea.type != TPM_ALG_KEYEDHASH)
        return TPM_RC_TYPE + RC_Unseal_itemHandle;
    if(   object->publicArea.objectAttributes.decrypt == SET
        || object->publicArea.objectAttributes.sign == SET
        || object->publicArea.objectAttributes.restricted == SET)
        return TPM_RC_ATTRIBUTES + RC_Unseal_itemHandle;

    // Command Output
    // Copy data
    MemoryCopy2B(&out->outData.b, &object->sensitive.sensitive.bits.b,
                 sizeof(out->outData.t.buffer));
    return TPM_RC_SUCCESS;
}
```
14.8 TPM2_ObjectChangeAuth

14.8.1 General Description

This command is used to change the authorization secret for a TPM-resident object. If successful, a new private area for the TPM-resident object associated with objectHandle is returned, which includes the new authorization value.

This command does not change the authorization of the TPM-resident object on which it operates. Therefore, the old authValue (of the TPM-resident object) is used when generating the response HMAC key if required.

NOTE 1 The returned outPrivate will need to be loaded before the new authorization will apply.

NOTE 2 The TPM-resident object may be persistent and changing the authorization value of the persistent object could prevent other users from accessing the object. This is why this command does not change the TPM-resident object.

EXAMPLE If a persistent key is being used as a Storage Root Key and the authorization of the key is a well-known value so that the key can be used generally, then changing the authorization value in the persistent key would deny access to other users.

This command may not be used to change the authorization value for an NV Index or a Primary Object.

NOTE 3 If an NV Index is to have a new authorization, it is done with TPM2_NV_ChangeAuth().

NOTE 4 If a Primary Object is to have a new authorization, it needs to be recreated (TPM2_CreatePrimary()).
### 14.8.2 Command and Response

#### Table 33 — TPM2_ObjectChangeAuth Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ObjectChangeAuth</td>
</tr>
<tr>
<td></td>
<td></td>
<td>handle of the object</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@objectHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: ADMIN</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>parentHandle</td>
<td>handle of the parent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_AUTH</td>
<td>newAuth</td>
<td>new authorization value</td>
</tr>
</tbody>
</table>

#### Table 34 — TPM2_ObjectChangeAuth Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_PRIVATE</td>
<td>outPrivate</td>
<td>private area containing the new authorization value</td>
</tr>
</tbody>
</table>
#include "InternalRoutines.h"
#include "ObjectChangeAuth_fp.h"
#include "Object_spt_fp.h"

## Error Returns

<table>
<thead>
<tr>
<th>Code</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIZE</td>
<td><code>newAuth</code> is larger than the size of the digest of the Name algorithm of <code>objectHandle</code></td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>the key referenced by <code>parentHandle</code> is not the parent of the object referenced by <code>objectHandle</code> or <code>objectHandle</code> is a sequence object.</td>
</tr>
</tbody>
</table>

```c
TPM2B_NAME objectQN, QNCompare;
TPM2B_NAME parentQN;
```

## Error Returns

- **TPM_RC_SIZE**: `newAuth` is larger than the size of the digest of the Name algorithm of `objectHandle`
- **TPM_RC_TYPE**: The key referenced by `parentHandle` is not the parent of the object referenced by `objectHandle` or `objectHandle` is a sequence object.

```c
// Input Validation
// Get object pointer
object = ObjectGet(in->objectHandle);

// Can not change auth on sequence object
if (ObjectIsSequence(object))
    return TPM_RC_TYPE + RC_ObjectChangeAuth_objectHandle;

// Make sure that the auth value is consistent with the nameAlg
if (MemoryRemoveTrailingZeros(&in->newAuth) > CryptGetHashDigestSize(object->publicArea.nameAlg))
    return TPM_RC_SIZE + RC_ObjectChangeAuth_newAuth;

// Check parent for object
// parent handle must be the parent of object handle. In this implementation we verify this by checking the QN of object. Other implementation may choose different method to verify this attribute.
ObjectGetQualifiedName(in->parentHandle, &parentQN);
ObjectComputerQualifiedName(&parentQN, object->publicArea.nameAlg, &object->name, &QNCompare);
ObjectGetQualifiedName(in->objectHandle, &objectQN);
if (!Memory2BEqual(&objectQN.b, &QNCompare.b))
    return TPM_RC_TYPE + RC_ObjectChangeAuth_parentHandle;
```

## Command Output

- **Copy internal sensitive area**
- **Copy authValue**
- **Prepare output private data from sensitive**

```c
SensitiveToPrivate(&sensitive, &object->name, in->parentHandle,
object->publicArea.nameAlg,
&out->outPrivate);

return TPM_RC_SUCCESS;

}
15 Duplication Commands

15.1 TPM2_Duplicate

15.1.1 General Description

This command duplicates a loaded object so that it may be used in a different hierarchy. The new parent key for the duplicate may be on the same or different TPM or TPM_RH_NULL. Only the public area of newParentHandle is required to be loaded.

**NOTE 1** Since the new parent may only be extant on a different TPM, it is likely that the new parent’s sensitive area could not be loaded in the TPM from which objectHandle is being duplicated.

If encryptedDuplication is SET in the object being duplicated, then the TPM shall return TPM_RC_SYMMETRIC if symmetricAlg is TPM_RH_NULL or TPM_RC_HIERARCHY if newParentHandle is TPM_RH_NULL.

The authorization for this command shall be with a policy session.

If fixedParent of objectHandle→attributes is SET, the TPM shall return TPM_RC_ATTRIBUTES. If objectHandle→nameAlg is TPM_ALG_NULL, the TPM shall return TPM_RC_TYPE.

The policySession→commandCode parameter in the policy session is required to be TPM_CC_Duplicate to indicate that authorization for duplication has been provided. This indicates that the policy that is being used is a policy that is for duplication, and not a policy that would approve another use. That is, authority to use an object does not grant authority to duplicate the object.

The policy is likely to include cpHash in order to restrict where duplication can occur. If TPM2_PolicyCpHash() has been executed as part of the policy, the policySession→cpHash is compared to the cpHash of the command.

If TPM2_PolicyDuplicationSelect() has been executed as part of the policy, the policySession→nameHash is compared to

\[ H_{\text{policyAlg}}(\text{objectHandle} \rightarrow \text{Name} || \text{newParentHandle} \rightarrow \text{Name}) \]  

If the compared hashes are not the same, then the TPM shall return TPM_RC_POLICY_FAIL.

**NOTE 2** It is allowed that policySession→nameHash and policySession→cpHash share the same memory space.

**NOTE 3** A duplication policy is not required to have either TPM2_PolicyDuplicationSelect() or TPM2_PolicyCpHash() as part of the policy. If neither is present, then the duplication policy may be satisfied with a policy that only contains TPM2_PolicyCommandCode(code = TPM_CC_Duplicate).

The TPM shall follow the process of encryption defined in the “Duplication” subclause of “Protected Storage Hierarchy” in Part 1 of this specification.
## 15.1.2 Command and Response

### Table 35 — TPM2_Duplicate Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Duplicate</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@objectHandle</td>
<td>loaded object to duplicate Auth Index: 1 Auth Role: DUP</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>newParentHandle</td>
<td>shall reference the public area of an asymmetric key Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>encryptionKeyIn</td>
<td>optional symmetric encryption key The size for this key is set to zero when the TPM is to generate the key. This parameter may be encrypted.</td>
</tr>
<tr>
<td>TPMT_SYM_DEF_OBJECT+</td>
<td>symmetricAlg</td>
<td>definition for the symmetric algorithm to be used for the inner wrapper may be TPM_ALG_NULL if no inner wrapper is applied</td>
</tr>
</tbody>
</table>

### Table 36 — TPM2_Duplicate Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>encryptionKeyOut</td>
<td>If the caller provided an encryption key or if symmetricAlg was TPM_ALG_NULL, then this will be the Empty Buffer; otherwise, it shall contain the TPM-generated, symmetric encryption key for the inner wrapper.</td>
</tr>
<tr>
<td>TPM2B_PRIVATE</td>
<td>duplicate</td>
<td>private area that may be encrypted by encryptionKeyIn; and may be doubly encrypted</td>
</tr>
<tr>
<td>TPM2B_ENCRYPTED_SECRET</td>
<td>outSymSeed</td>
<td>seed protected by the asymmetric algorithms of new parent (NP)</td>
</tr>
</tbody>
</table>
15.1.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Duplicate_fp.h"
#include "Object_spt_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>key to duplicate has fixedParent SET</td>
</tr>
<tr>
<td>TPM_RC_HIERARCHY</td>
<td>encryptedDuplication is SET and newParentHandle specifies Null Hierarchy</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>newParentHandle references invalid ECC key (public point not on the curve)</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>input encryption key size does not match the size specified in symmetric algorithm</td>
</tr>
<tr>
<td>TPM_RC_SYMMETRIC</td>
<td>encryptedDuplication is SET but no symmetric algorithm is provided</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>newParentHandle is neither a storage key nor TPM_RH_NULL; or the object has a NULL nameAlg</td>
</tr>
</tbody>
</table>

```c
TPM_RC

TPM2_Duplicate(
    Duplicate_In   *in,   // IN: input parameter list
    Duplicate_Out  *out   // OUT: output parameter list
)
{
    TPM_RC                  result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE          sensitive;

    UINT16                  innerKeySize = 0; // encrypt key size for inner wrap

    OBJECT                  *object;
    TPM2B_DATA              data;

    // Input Validation
    // Get duplicate object pointer
    object = ObjectGet(in->objectHandle);

    // duplicate key must have fixParent bit CLEAR.
    if(object->publicArea.objectAttributes.fixedParent == SET)
        return TPM_RC_ATTRIBUTES + RC_Duplicate_objectHandle;

    // Do not duplicate object with NULL nameAlg
    if(object->publicArea.nameAlg == TPM_ALG_NULL)
        return TPM_RC_TYPE + RC_Duplicate_newParentHandle;

    // new parent key must be a storage object or TPM_RH_NULL
    if(in->newParentHandle != TPM_RH_NULL
        && !ObjectIsStorage(in->newParentHandle))
        return TPM_RC_TYPE + RC_Duplicate_newParentHandle;

    // If the duplicates object has encryptedDuplication SET, then there must be
    // an inner wrapper and the new parent may not be TPM_RH_NULL
    if(object->publicArea.objectAttributes.encryptedDuplication == SET)
    {
        if(in->symmetricAlg.algorithm == TPM_ALG_NULL)
            return TPM_RC_SYMMETRIC + RC_Duplicate_symmetricAlg;
        if(in->newParentHandle == TPM_RH_NULL)
```
return TPM_RC_HIERARCHY + RC_Duplicate_newParentHandle;
}

if(in->symmetricAlg.algorithm == TPM_ALG_NULL) {
    // if algorithm is TPM_ALG_NULL, input key size must be 0
    if(in->encryptionKeyIn.t.size != 0)
        return TPM_RC_SIZE + RC_Duplicate_encryptionKeyIn;
} else {
    // Get inner wrap key size
    innerKeySize = in->symmetricAlg.keyBits.sym;
    // If provided the input symmetric key must match the size of the algorithm
    if(in->encryptionKeyIn.t.size != 0 && in->encryptionKeyIn.t.size != (innerKeySize + 7) / 8)
        return TPM_RC_SIZE + RC_Duplicate_encryptionKeyIn;
}

// Command Output

if(in->newParentHandle != TPM_RH_NULL) {
    // Make encrypt key and its associated secret structure. A TPM_RC_KEY error may be returned at this point
    out->outSymSeed.t.size = sizeof(out->outSymSeed.t.secret);
    result = CryptSecretEncrypt(in->newParentHandle,
        "DUPLICATE", &data, &out->outSymSeed);
    pAssert(result != TPM_RC_VALUE);
    if(result != TPM_RC_SUCCESS)
        return result;
} else {
    // Do not apply outer wrapper
    data.t.size = 0;
    out->outSymSeed.t.size = 0;
}

// Copy sensitive area
sensitive = object->sensitive;

// Prepare output private data from sensitive
SensitiveToDuplicate(&sensitive, &object->name, in->newParentHandle,
    object->publicArea.nameAlg, (TPM2B_SEED *) &data,
    &in->symmetricAlg, &in->encryptionKeyIn,
    &out->duplicate);

out->encryptionKeyOut = in->encryptionKeyIn;
return TPM_RC_SUCCESS;
15.2 TPM2_Rewrap

15.2.1 General Description

This command allows the TPM to serve in the role as a Duplication Authority. If proper authorization for use of the oldParent is provided, then an HMAC key and a symmetric key are recovered from inSymSeed and used to integrity check and decrypt inDuplicate. A new protection seed value is generated according to the methods appropriate for newParent and the blob is re-encrypted and a new integrity value is computed. The re-encrypted blob is returned in outDuplicate and the symmetric key returned in outSymKey.

In the rewrap process, L is “DUPLICATE” (see “Terms and Definitions” in Part 1).

If inSymSeed has a zero length, then oldParent is required to be TPM_RH_NULL and no decryption of inDuplicate takes place.

If newParent is TPM_RH_NULL, then no encryption is performed on outDuplicate. outSymSeed will have a zero length. See Part 2 encryptedDuplication.
### 15.2.2 Command and Response

#### Table 37 — TPM2_Rewrap Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Rewrap</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>@oldParent</td>
<td>parent of object, Auth Index: 1, Auth Role: User</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>newParent</td>
<td>new parent of the object, Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_PRIVATE</td>
<td>inDuplicate</td>
<td>an object encrypted using symmetric key derived from inSymSeed</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>name</td>
<td>the Name of the object being rewrapped</td>
</tr>
<tr>
<td>TPM2B_ENCRYPTED_SECRET</td>
<td>inSymSeed</td>
<td>seed for symmetric key needs oldParent private key to recover the seed and generate the symmetric key</td>
</tr>
</tbody>
</table>

#### Table 38 — TPM2_Rewrap Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_PRIVATE</td>
<td>outDuplicate</td>
<td>an object encrypted using symmetric key derived from outSymSeed</td>
</tr>
<tr>
<td>TPM2B_ENCRYPTED_SECRET</td>
<td>outSymSeed</td>
<td>seed for a symmetric key protected by newParent asymmetric key</td>
</tr>
</tbody>
</table>
15.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Rewrap_fp.h"
#include "Object_spt_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td><code>newParent</code> is not a decryption key</td>
</tr>
<tr>
<td>TPM_RC_HANDLE</td>
<td><code>oldParent</code> does not consistent with <code>inSymSeed</code></td>
</tr>
<tr>
<td>TPM_RC_INTEGRITY</td>
<td>the integrity check of <code>inDuplicate</code> failed</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>for an ECC key, the public key is not on the curve of the curve ID</td>
</tr>
<tr>
<td>TPM_RC_KEY_SIZE</td>
<td>the decrypted input symmetric key size does not matches the symmetric algorithm key size of <code>oldParent</code></td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td><code>oldParent</code> is not a storage key, or <code>newParent</code> is not a storage key</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>for an <code>oldParent</code>; RSA key, the data to be decrypted is greater than the public exponent</td>
</tr>
<tr>
<td>Unmarshal errors</td>
<td>errors during unmarshaling the input encrypted buffer to a ECC public key, or unmarshal the private buffer to sensitive</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_Rewrap(
    TPM2_Rewrap_In* in, // IN: input parameter list
    TPM2_Rewrap_Out* out // OUT: output parameter list
)
{
    TPM_RC result = TPM_RC_SUCCESS;
    OBJECT* oldParent;
    TPM2B_DATA data; // symmetric key
    UINT16 hashSize = 0;
    TPM2B_PRIVATE privateBlob; // A temporary private blob

    // Input Validation
    if((in->inSymSeed.t.size == 0 && in->oldParent != TPM_RH_NULL) || (in->inSymSeed.t.size != 0 && in->oldParent == TPM_RH_NULL))
        return TPM_RC_HANDLE + RC_Rewrap_oldParent;

    if(in->oldParent != TPM_RH_NULL)
    {
        // Get old parent pointer
        oldParent = ObjectGet(in->oldParent);

        // old parent key must be a storage object
        if(!ObjectIsStorage(in->oldParent))
            return TPM_RC_TYPE + RC_Rewrap_oldParent;

        // Decrypt input secret data via asymmetric decryption. A
        // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
        // point
        result = CryptSecretDecrypt(in->oldParent, NULL, "DUPLICATE", &in->inSymSeed, &data);

        if(result != TPM_RC_SUCCESS)
            return TPM_RC_VALUE + RC_Rewrap_inSymSeed;
    }
```

```c
```
// Unwrap Outer
result = UnwrapOuter(in->oldParent, &in->name,
    oldParent->publicArea.nameAlg, (TPM2B_SEED *) &data,
    FALSE,
in->inDuplicate.t.size, in->inDuplicate.t.buffer);
if(result != TPM_RC_SUCCESS)
    return RcSafeAddToResult(result, RC_Rewrap_inDuplicate);

// Copy unwrapped data to temporary variable, remove the integrity field
hashSize = sizeof(UINT16) +
    CryptGetHashDigestSize(oldParent->publicArea.nameAlg);
privateBlob.t.size = in->inDuplicate.t.size - hashSize;
MemoryCopy(privateBlob.t.buffer, in->inDuplicate.t.buffer + hashSize,
    privateBlob.t.size, sizeof(privateBlob.t.buffer));
} else {
    // No outer wrap from input blob. Direct copy.
    privateBlob = in->inDuplicate;
}

if(in->newParent != TPM_RH_NULL) {
    OBJECT   *newParent;
    newParent = ObjectGet(in->newParent);

    // New parent must be a storage object
    if(!ObjectIsStorage(in->newParent))
        return TPM_RC_TYPE + RC_Rewrap_newParent;

    // Make new encrypt key and its associated secret structure. A
    // TPM_RC_VALUE error may be returned at this point if RSA algorithm is
    // enabled in TPM
    out->outSymSeed.t.size = sizeof(out->outSymSeed.t.secret);
    result = CryptSecretEncrypt(in->newParent,
        "DUPLICATE", &data, &out->outSymSeed);
    if(result != TPM_RC_SUCCESS) return result;

    // Command output
    // Copy temporary variable to output, reserve the space for integrity
    hashSize = sizeof(UINT16) +
        CryptGetHashDigestSize(newParent->publicArea.nameAlg);
    out->outDuplicate.t.size = privateBlob.t.size;
    MemoryCopy(out->outDuplicate.t.buffer + hashSize, privateBlob.t.buffer,
        privateBlob.t.size, sizeof(out->outDuplicate.t.buffer));

    // Produce outer wrapper for output
    out->outDuplicate.t.size = ProduceOuterWrap(in->newParent, &in->name,
        newParent->publicArea.nameAlg,
        (TPM2B_SEED *) &data,
        FALSE,
        out->outDuplicate.t.size,
        out->outDuplicate.t.buffer);
} else // New parent is a null key so there is no seed
{
    out->outSymSeed.t.size = 0;

    // Copy privateBlob directly
    out->outDuplicate = privateBlob;
}
return TPM_RC_SUCCESS;
}
15.3 TPM2_Import

15.3.1 General Description

This command allows an object to be encrypted using the symmetric encryption values of a Storage Key. After encryption, the object may be loaded and used in the new hierarchy. The imported object (duplicate) may be singly encrypted, multiply encrypted, or unencrypted.

If fixedTPM or fixedParent is SET in objectPublic, the TPM shall return TPM_RC_ATTRIBUTES.

If encryptedDuplication is SET in the object referenced by parentHandle, then encryptedDuplication shall be set in objectPublic (TPM_RC_ATTRIBUTES). However, see Note 2.

Recovery of the sensitive data of the object occurs in the TPM in a three-step process in the following order:

- If present, the outer layer of symmetric encryption is removed. If inSymSeed has a non-zero size, the asymmetric parameters and private key of parentHandle are used to recover the seed used in the creation of the HMAC key and encryption keys used to protect the duplication blob. When recovering the seed, L is "DUPLICATE".

  NOTE 1  If the encryptedDuplication attribute of the object is SET, the TPM shall return TPM_RC_ATTRIBUTES if inSymSeed is an empty buffer.

- If present, the inner layer of symmetric encryption is removed. If encryptionKey and symmetricAlg are provided, they are used to decrypt duplication.

- If present, the integrity value of the blob is checked. The presence of the integrity value is indicated by a non-zero value for duplicate.data.integrity.size. The integrity of the private area is validated using the Name of objectPublic in the integrity HMAC computation. If either the outer layer or inner layer of encryption is performed, then the integrity value shall be present.

If the inner or outer wrapper is present, then a valid integrity value shall be present or the TPM shall return TPM_RC_INTEGRITY.

NOTE 2  It is not necessary to validate that the sensitive area data is cryptographically bound to the public area other than that the Name of the public area is included in the HMAC. However, if the binding is not validated by this command, the binding must be checked each time the object is loaded. For an object that is imported under a parent with fixedTPM SET, binding need only be checked at import. If the parent has fixedTPM CLEAR, then the binding needs to be checked each time the object is loaded, or before the TPM performs an operation for which the binding affects the outcome of the operation (for example, TPM2_PolicySigned() or TPM2_Certify()).

Similarly, if the new parent's fixedTPM is set, the encryptedDuplication state need only be checked at import.

If the new parent is not fixedTPM, then that object will be loadable on any TPM (including SW versions) on which the new parent exists. This means that, each time an object is loaded under a parent that is not fixedTPM, it is necessary to validate all of the properties of that object. If the parent is fixedTPM, then the new private blob is integrity protected by the TPM that "owns" the parent. So, it is sufficient to validate the object’s properties (attribute and public-private binding) on import and not again.

Before duplicate.buffer is decrypted using the symmetric key of the parent, the integrity value shall be checked before the sensitive area is used, or unmarshaled.

After integrity checks and decryption, the TPM will create a new symmetrically encrypted private area using the encryption key of the parent.

NOTE 3  Checking the integrity before the data is used prevents attacks on the sensitive area by fuzzing the data and looking at the differences in the response codes.
NOTE 4  The symmetric re-encryption is the normal integrity generation and symmetric encryption applied to a child object.
### 15.3.2 Command and Response

#### Table 39 — TPM2_Import Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Import</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@parentHandle</td>
<td>the handle of the new parent for the object</td>
</tr>
<tr>
<td></td>
<td>Auth Index</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Auth Role</td>
<td>USER</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>encryptionKey</td>
<td>the optional symmetric encryption key used as the inner wrapper for <code>duplicate</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If <code>symmetricAlg</code> is TPM_ALG_NULL, then this parameter shall be the Empty Buffer.</td>
</tr>
<tr>
<td>TPM2B_PUBLIC</td>
<td>objectPublic</td>
<td>the public area of the object to be imported</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This is provided so that the integrity value for <code>duplicate</code> and the object attributes can be checked.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE Even if the integrity value of the object is not checked on input, the object Name is required to create the integrity value for the imported object.</td>
</tr>
<tr>
<td>TPM2B_PRIVATE</td>
<td>duplicate</td>
<td>the symmetrically encrypted duplicate object that may contain an inner symmetric wrapper</td>
</tr>
<tr>
<td>TPM2B_ENCRYPTED_SECRET</td>
<td>inSymSeed</td>
<td>symmetric key used to encrypt <code>duplicate</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>inSymSeed</code> is encrypted/encoded using the algorithms of <code>newParent</code>.</td>
</tr>
<tr>
<td>TPMT_SYM_DEF_OBJECT+</td>
<td>symmetricAlg</td>
<td>definition for the symmetric algorithm to use for the inner wrapper</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If this algorithm is TPM_ALG_NULL, no inner wrapper is present and <code>encryptionKey</code> shall be the Empty Buffer.</td>
</tr>
</tbody>
</table>

#### Table 40 — TPM2_Import Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_PRIVATE</td>
<td>outPrivate</td>
<td>the sensitive area encrypted with the symmetric key of <code>parentHandle</code></td>
</tr>
</tbody>
</table>
### 15.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Import_fp.h"
#include "Object_spt_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ASYMMETRIC</td>
<td>non-duplicable storage key represented by <code>objectPublic</code> and its parent referenced by <code>parentHandle</code> have different public params</td>
</tr>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>attributes <code>FixedTPM</code> and <code>fixedParent</code> of <code>objectPublic</code> are not both CLEAR; or <code>inSymSeed</code> is nonempty and <code>parentHandle</code> does not reference a decryption key; or <code>objectPublic</code> and <code>parentHandle</code> have incompatible or inconsistent attributes</td>
</tr>
<tr>
<td>TPM_RC_BINDING</td>
<td><code>duplicate</code> and <code>objectPublic</code> are not cryptographically bound</td>
</tr>
<tr>
<td>TPM_RC_ECC_POINT</td>
<td><code>inSymSeed</code> is nonempty and ECC point in <code>inSymSeed</code> is not on the curve</td>
</tr>
<tr>
<td>TPM_RC_HASH</td>
<td>non-duplicable storage key represented by <code>objectPublic</code> and its parent referenced by <code>parentHandle</code> have different name algorithm</td>
</tr>
<tr>
<td>TPM_RC_INSUFFICIENT</td>
<td><code>inSymSeed</code> is nonempty and failed to retrieve ECC point from the secret; or unmarshaling sensitive value from <code>duplicate</code> failed is the result of <code>inSymSeed</code> decryption</td>
</tr>
<tr>
<td>TPM_RC_INTEGRITY</td>
<td><code>duplicate</code> integrity is broken</td>
</tr>
<tr>
<td>TPM_RC_KDF</td>
<td><code>objectPublic</code> representing decrypting keyed hash object specifies invalid KDF</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>inconsistent parameters of <code>objectPublic</code>; or <code>inSymSeed</code> is nonempty and <code>parentHandle</code> does not reference a key of supported type; or invalid key size in <code>objectPublic</code> representing an asymmetric key</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td><code>inSymSeed</code> is nonempty and multiplication resulted in ECC point at infinity</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>no available object slot</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>inconsistent attributes <code>decrypt</code>, <code>sign</code>, <code>restricted</code> and key’s scheme ID in <code>objectPublic</code>; or hash algorithm is inconsistent with the scheme ID for keyed hash object</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td><code>authPolicy</code> size does not match digest size of the name algorithm in <code>objectPublic</code>; or <code>symmetricAlg</code> and <code>encryptionKey</code> have different sizes; or <code>inSymSeed</code> is nonempty and it is not of the same size as RSA key referenced by <code>parentHandle</code>; or unmarshaling sensitive value from <code>duplicate</code> failed</td>
</tr>
<tr>
<td>TPM_RC_SYMMETRIC</td>
<td><code>objectPublic</code> is either a storage key with no symmetric algorithm or a non-storage key with symmetric algorithm different from <code>TPM_ALG_NULL</code></td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>unsupported type of <code>objectPublic</code>; or non-duplicable storage key represented by <code>objectPublic</code> and its parent referenced by <code>parentHandle</code> are of different types; or <code>parentHandle</code> is not a storage key; or only the public portion of <code>parentHandle</code> is loaded; or <code>objectPublic</code> and <code>duplicate</code> are of different types</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>nonempty <code>inSymSeed</code> and its numeric value is greater than the modulus of the key referenced by <code>parentHandle</code> or <code>inSymSeed</code> is larger than the size of the digest produced by the name algorithm of the symmetric key referenced by <code>parentHandle</code></td>
</tr>
</tbody>
</table>
\textbf{TPM2\_Import(}
\begin{verbatim}
    Import_In  *in,  // IN: input parameter list
    Import_Out *out   // OUT: output parameter list
\end{verbatim}
\})

\begin{verbatim}
TPM_RC                  result = TPM_RC_SUCCESS;
OBJECT                  *parentObject;
TPM2B_DATA              data;       // symmetric key
TPMT_SENSITIVE          sensitive;
TPM2B_NAME              name;
UINT16                  innerKeySize = 0;     // encrypt key size for inner
                         // wrapper

// Input Validation

// FixedTPM and fixedParent must be CLEAR
if (   in->objectPublic.t.publicArea.objectAttributes.fixedTPM == SET
     || in->objectPublic.t.publicArea.objectAttributes.fixedParent == SET)
    return TPM_RC_ATTRIBUTES + RC_Import_objectPublic;

// Get parent pointer
parentObject = ObjectGet(in->parentHandle);

if (!AreAttributesForParent(parentObject))
    return TPM_RC_TYPE + RC_Import_parentHandle;

if (in->symmetricAlg.algorithm != TPM_ALG_NULL)
{
    // Get inner wrap key size
    innerKeySize = in->symmetricAlg.keyBits.sym;
    // Input symmetric key must match the size of algorithm.
    if (in->encryptionKey.t.size != (innerKeySize + 7) / 8)
        return TPM_RC_SIZE + RC_Import_encryptionKey;
}
else
{
    // If input symmetric algorithm is NULL, input symmetric key size must
    // be 0 as well
    if (in->encryptionKey.t.size != 0)
        return TPM_RC_SIZE + RC_Import_encryptionKey;
}

// See if there is an outer wrapper
if (in->inSymSeed.t.size != 0)
{
    // Decrypt input secret data via asymmetric decryption. TPM_RC_ATTRIBUTES,
    // TPM_RC_ECC_POINT, TPM_RC_INSUFFICIENT, TPM_RC_KEY, TPM_RC_NO_RESULT,
    // TPM_RC_SIZE, TPM_RC_VALUE may be returned at this point
    result = CryptSecretDecrypt(in->parentHandle, NULL, "DUPLICATE",
                          &in->inSymSeed, &data);
    pAssert(result != TPM_RC_BINDING);
    if (result != TPM_RC_SUCCESS)
        return TPM_RC_VALUE + RC_Import_inSymSeed;
}
else
{
    data.t.size = 0;
}

// Compute name of object
ObjectComputeName(&(in->objectPublic.t.publicArea), &name);
\end{verbatim}
// Retrieve sensitive from private.
// TPM_RC_INSUFFICIENT, TPM_RC_INTEGRITY, TPM_RC_SIZE may be returned here.
result = DuplicateToSensitive(&in->duplicate, &name, in->parentHandle,
in->objectPublic.t.publicArea.nameAlg,
(TPM2B_SEED *) &data, &in->symmetricAlg,
&in->encryptionKey, &sensitive);
if(result != TPM_RC_SUCCESS)
  return RcSafeAddToResult(result, RC_Import_duplicate);

// If the parent of this object has fixedTPM SET, then fully validate this
// object so that validation can be skipped when it is loaded
if(appendToObject->publicArea.objectAttributes.fixedTPM == SET)
{
  TPM_HANDLE       objectHandle;

  // Perform self check on input public area. A TPM_RC_SIZE, TPM_RC_SCHEME,
  // TPM_RC_VALUE, TPM_RC_SYMMETRIC, TPM_RC_TYPE, TPM_RC_HASH,
  // TPM_RC_ASYMMETRIC, TPM_RC_ATTRIBUTES or TPM_RC_KDF error may be returned
  // at this point
  result = PublicAttributesValidation(TRUE, in->parentHandle,
                                       &in->objectPublic.t.publicArea);
  if(result != TPM_RC_SUCCESS)
    return RcSafeAddToResult(result, RC_Import_objectPublic);

  // Create internal object. A TPM_RC_KEY_SIZE, TPM_RC_KEY or
  // TPM_RC_OBJECT_MEMORY error may be returned at this point
  result = ObjectLoad(TPM_RH_NULL, &in->objectPublic.t.publicArea,
                      &sensitive, NULL, in->parentHandle, FALSE,
                      &objectHandle);
  if(result != TPM_RC_SUCCESS)
    return result;

  // Don't need the object, just needed the checks to be performed so
  // flush the object
  ObjectFlush(objectHandle);
}

// Command output

// Prepare output private data from sensitive
SensitiveToPrivate(&sensitive, &name, in->parentHandle,
in->objectPublic.t.publicArea.nameAlg,
&out->outPrivate);
return TPM_RC_SUCCESS;
16 Asymmetric Primitives

16.1 Introduction

The commands in this clause provide low-level primitives for access to the asymmetric algorithms implemented in the TPM. Many of these commands are only allowed if the asymmetric key is an unrestricted key.

16.2 TPM2_RSA_Encrypt

16.2.1 General Description

This command performs RSA encryption using the indicated padding scheme according to PKCS#1v2.1 (PKCS#1). If the scheme of keyHandle is TPM_ALG_NULL, then the caller may use inScheme to specify the padding scheme. If scheme of keyHandle is not TPM_ALG_NULL, then inScheme shall either be TPM_ALG_NULL or be the same as scheme (TPM_RC_SCHEME).

The key referenced by keyHandle is required to be an RSA key (TPM_RC_KEY) with the decrypt attribute SET (TPM_RC_ATTRIBUTES).

NOTE Requiring that the decrypt attribute be set allows the TPM to ensure that the scheme selection is done with the presumption that the scheme of the key is a decryption scheme selection. It is understood that this command will operate on a key with only the public part loaded so the caller may modify any key in any desired way. So, this constraint only serves to simplify the TPM logic.

The three types of allowed padding are:

1) TPM_ALG_OAEP – Data is OAEP padded as described in 7.1 of PKCS#1v2.1. The only supported mask generation is MGF1.
2) TPM_ALG_RSAES – Data is padded as described in 7.2 of PKCS#1v2.1.
3) TPM_ALG_NULL – Data is not padded by the TPM and the TPM will treat message as an unsigned integer and perform a modular exponentiation of message using the public exponent of the key referenced by keyHandle. This scheme is only used if both the scheme in the key referenced by keyHandle is TPM_ALG_NULL, and the inScheme parameter of the command is TPM_ALG_NULL. The input value cannot be larger than the public modulus of the key referenced by keyHandle.

<table>
<thead>
<tr>
<th>keyHandle→scheme</th>
<th>inScheme</th>
<th>padding scheme used</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ALG_NULL</td>
<td>TPM_ALG_NULL</td>
<td>none</td>
</tr>
<tr>
<td></td>
<td>TPM_ALG_RSAES</td>
<td>RSAES</td>
</tr>
<tr>
<td></td>
<td>TPM_ALG_OAEP</td>
<td>OAEP</td>
</tr>
<tr>
<td>TPM_ALG_RSAES</td>
<td>TPM_ALG_NULL</td>
<td>RSAES</td>
</tr>
<tr>
<td></td>
<td>TPM_ALG_RSAES</td>
<td>RSAES</td>
</tr>
<tr>
<td></td>
<td>TPM_ALG_OAEP</td>
<td>error (TPM_RC_SCHEME)</td>
</tr>
<tr>
<td>TPM_ALG_OAEP</td>
<td>TPM_ALG_NULL</td>
<td>OAEP</td>
</tr>
<tr>
<td></td>
<td>TPM_ALG_RSAES</td>
<td>error (TPM_RC_SCHEME)</td>
</tr>
<tr>
<td></td>
<td>TPM_AGL_OAEP</td>
<td>OAEP</td>
</tr>
</tbody>
</table>
After padding, the data is RSAEP encrypted according to 5.1.1 of PKCS#1v2.1.

NOTE 1  It is required that *decrypt* be SET so that the commands that load a key can validate that the scheme is consistent rather than have that deferred until the key is used.

NOTE 2  If it is desired to use a key that had restricted SET, the caller may CLEAR restricted and load the public part of the key and use that unrestricted version of the key for encryption.

If *inScheme* is used, and the scheme requires a hash algorithm it may not be TPM_ALG_NULL.

NOTE 3  Because only the public portion of the key needs to be loaded for this command, the caller can manipulate the attributes of the key in any way desired. As a result, the TPM shall not check the consistency of the attributes. The only property checking is that the key is an RSA key and that the padding scheme is supported.

The *message* parameter is limited in size by the padding scheme according to the following table:

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Maximum Message Length (mLen) in Octets</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ALG_OAEP</td>
<td>mLen ≤ k – 2hLen – 2</td>
<td></td>
</tr>
<tr>
<td>TPM_ALG_RSAES</td>
<td>mLen ≤ k – 11</td>
<td></td>
</tr>
<tr>
<td>TPM_ALG_NULL</td>
<td>mLen ≤ k</td>
<td>The numeric value of the message must be less than the numeric value of the public modulus (n).</td>
</tr>
</tbody>
</table>

The numeric value of the message must be less than the numeric value of the public modulus (n).

NOTES
1)  k := the number of bytes in the public modulus
2)  hLen := the number of octets in the digest produced by the hash algorithm used in the process

The *label* parameter is optional. If provided (*label.size != 0*) then the TPM shall return TPM_RC_VALUE if the last octet in *label* is not zero. If a zero octet occurs before *label.buffer[.label.size-1]*, the TPM shall truncate the label at that point. The terminating octet of zero is included in the *label* used in the padding scheme.

NOTE 4  If the scheme does not use a label, the TPM will still verify that label is properly formatted if label is present.

The function returns padded and encrypted value *outData*.

The *message* parameter in the command may be encrypted using parameter encryption.

NOTE 5  Only the public area of *keyHandle* is required to be loaded. A public key may be loaded with any desired scheme. If the scheme is to be changed, a different public area must be loaded.
### 16.2.2 Command and Response

#### Table 43 — TPM2_RSA_Encrypt Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_RSA_Encrypt</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>keyHandle</td>
<td>reference to public portion of RSA key to use for encryption</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_PUBLIC_KEY_RSA</td>
<td>message</td>
<td>message to be encrypted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE 1 The data type was chosen because it limits the overall size of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>input to no greater than the size of the largest RSA public key. This may</td>
</tr>
<tr>
<td></td>
<td></td>
<td>be larger than allowed for keyHandle.</td>
</tr>
<tr>
<td>TPMT_RSA_DECRYPT+</td>
<td>inScheme</td>
<td>the padding scheme to use if scheme associated with keyHandle is TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>label</td>
<td>optional label L to be associated with the message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size of the buffer is zero if no label is present</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE 2 See description of label above.</td>
</tr>
</tbody>
</table>

#### Table 44 — TPM2_RSA_Encrypt Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_PUBLIC_KEY_RSA</td>
<td>outData</td>
<td>encrypted output</td>
</tr>
</tbody>
</table>
16.2.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "RSA_Encrypt_fp.h"
#ifdef TPM_ALG_RSA

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>decrypt attribute is not SET in key referenced by keyHandle</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>keyHandle does not reference an RSA key</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>incorrect input scheme, or the chosen scheme is not a valid RSA decrypt scheme</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>the numeric value of message is greater than the public modulus of the key referenced by keyHandle, or label is not a null-terminated string</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_RSA_Encrypt(
    RSA_Encrypt_In *in,  // IN: input parameter list
    RSA_Encrypt_Out *out  // OUT: output parameter list
)
{
    TPM_RC                  result;
    OBJECT                  *rsaKey;
    TPM2_RSA_DECRYPT       *scheme;
    char                    *label = NULL;

    // Input Validation
    rsaKey = ObjectGet(in->keyHandle);

    // selected key must be an RSA key
    if(rsaKey->publicArea.type != TPM_ALG_RSA)
        return TPM_RC_KEY + RC_RSA_Encrypt_keyHandle;

    // selected key must have the decryption attribute
    if(rsaKey->publicArea.objectAttributes.decrypt != SET)
        return TPM_RC_ATTRIBUTES + RC_RSA_Encrypt_keyHandle;

    // Is there a label?
    if(in->label.t.size > 0)
        {  // label is present, so make sure that it is NULL-terminated
            if(in->label.t.buffer[in->label.t.size - 1] != 0)
                return TPM_RC_VALUE + RC_RSA_Encrypt_label;
            label = (char *)in->label.t.Buffer;
        }

    // Command Output
    scheme = CryptSelectRSAScheme(in->keyHandle, &in->inScheme);
    if(scheme == NULL)
        return TPM_RC_SCHEME + RC_RSA_Encrypt_inScheme;

    // Encryption. TPM_RC_VALUE, or TPM_RC_SCHEME errors my be returned but
    // CryptEncryptRSA. Note: It can also return TPM_RC_ATTRIBUTES if the key does not have the decrypt attribute but that was checked above.
    out->outData.t.size = sizeof(out->outData.t.buffer);
```
result = CryptEncryptRSA(&out->outData.t.size, out->outData.t.buffer, rsaKey, 
    scheme, in->message.t.size, in->message.t.buffer, 
    label);

    return result;
}

#endif
16.3 TPM2_RSA_Decrypt

16.3.1 General Description

This command performs RSA decryption using the indicated padding scheme according to PKCS#1v2.1 (PKCS#1).

The scheme selection for this command is the same as for TPM2_RSA_Encrypt() and is shown in Table 41.

The key referenced by keyHandle shall be an RSA key (TPM_RC_KEY) with restricted CLEAR and decrypt SET (TPM_RC_ATTRIBUTES).

This command uses the private key of keyHandle for this operation and authorization is required.

The TPM will perform a modular exponentiation of ciphertext using the private exponent associated with keyHandle (this is described in PKCS#1v2.1, clause 5.1.2). It will then validate the padding according to the selected scheme. If the padding checks fail, TPM_RC_VALUE is returned. Otherwise, the data is returned with the padding removed. If no padding is used, the returned value is an unsigned integer value that is the result of the modular exponentiation of cipherText using the private exponent of keyHandle.

The returned value may include leading octets zeros so that it is the same size as the public modulus. For the other padding schemes, the returned value will be smaller than the public modulus but will contain all the data remaining after padding is removed and this may include leading zeros if the original encrypted value contained leading zeros.

If a label is used in the padding process of the scheme, the label parameter is required to be present in the decryption process and label is required to be the same in both cases. The TPM shall verify that the label is consistent and if not it shall return TPM_RC_VALUE. If label is present (label.size != 0), it shall be a NULL-terminated string or the TPM will return TPM_RC_VALUE.

NOTE 1 The size of label includes the terminating null.

The message parameter in the response may be encrypted using parameter encryption.

If the decryption scheme does not require a hash function, the hash parameter of inScheme may be set to any valid hash function or TPM_ALG_NULL.

If the description scheme does not require a label, the value in label is not used but the size of the label field is checked for consistency with the indicated data type (TPM2B_DATA). That is, the field may not be larger than allowed for a TPM2B_DATA.
### 16.3.2 Command and Response

#### Table 45 — TPM2_RSA_Decrypt Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_RSA_Decrypt RSA key to use for decryption</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@keyHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td>Auth Role: USER</td>
<td></td>
</tr>
<tr>
<td>TPM2B_PUBLIC_KEY_RSA</td>
<td>cipherText</td>
<td>NOTE: An encrypted RSA data block is the size of the public modulus.</td>
</tr>
<tr>
<td>TPMT_RSA_DECRYPT+</td>
<td>inScheme</td>
<td>The padding scheme to use if scheme associated with keyHandle is TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>label</td>
<td>Label whose association with the message is to be verified</td>
</tr>
</tbody>
</table>

#### Table 46 — TPM2_RSA_Decrypt Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_PUBLIC_KEY_RSA</td>
<td>message</td>
<td>Decrypted output</td>
</tr>
</tbody>
</table>
16.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "RSA_Decrypt_fp.h"
#ifdef TPM_ALG_RSA

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>keyHandle does not reference an unrestricted decrypt key</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>incorrect input scheme, or the chosen scheme is not a valid RSA decrypt scheme</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>cipherText is not the size of the modulus of key referenced by keyHandle</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>label is not a null terminated string or the value of cipherText is greater that the modulus of keyHandle</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_RSA_Decrypt(
RSA_Decrypt_In  *in,          // IN: input parameter list
RSA_Decrypt_Out *out          // OUT: output parameter list
)
{
    TPM_RC                       result;
    OBJECT                      *rsaKey;
    TPMT_RSA_DECRYPT            *scheme;
    char                        *label = NULL;

    // Input Validation

    rsaKey = ObjectGet(in->keyHandle);

    // The selected key must be an RSA key
    if(rsaKey->publicArea.type != TPM_ALG_RSA)
        return TPM_RC_KEY + RC_RSA_Decrypt_keyHandle;

    // The selected key must be an unrestricted decryption key
    if(!rsaKey->publicArea.objectAttributes.restricted
        || rsaKey->publicArea.objectAttributes.decrypt == CLEAR)
        return TPM_RC_ATTRIBUTES + RC_RSA_Decrypt_keyHandle;

    // NOTE: Proper operation of this command requires that the sensitive area
    // of the key is loaded. This is assured because authorization is required
    // to use the sensitive area of the key. In order to check the authorization,
    // the sensitive area has to be loaded, even if authorization is with policy.

    // If label is present, make sure that it is a NULL-terminated string
    if(in->label.t.size > 0)
    {
        // Present, so make sure that it is NULL-terminated
        if(in->label.t.buffer[in->label.t.size - 1] != 0)
            return TPM_RC_VALUE + RC_RSA_Decrypt_label;
        label = (char *)in->label.t.buffer;
    }

    // Command Output

    // Select a scheme for decrypt.
    scheme = CryptSelectRSAScheme(in->keyHandle, &in->inScheme);
    if(scheme == NULL)
```

        return TPM_RC_SCHEME + RC_RSA_Decrypt_inScheme;

    // Decryption.  TPM_RC_VALUE, TPM_RC_SIZE, and TPM_RC_KEY error may be
    // returned by CryptDecryptRSA.
    // NOTE: CryptDecryptRSA can also return TPM_RC_ATTRIBUTES or TPM_RC_BINDING
    // when the key is not a decryption key but that was checked above.
    out->message.t.size = sizeof(out->message.t.buffer);
    result = CryptDecryptRSA(&out->message.t.size, out->message.t.buffer, rsaKey,
                              scheme, in->cipherText.t.size,
                              in->cipherText.t.buffer,
                              label);

    return result;
  }
#endif
16.4 TPM2_ECDH_KeyGen

16.4.1 General Description

This command uses the TPM to generate an ephemeral key pair \((d, Q_e)\) where \(Q_e := [d] G\). It uses the private ephemeral key and a loaded public key \((Q_S)\) to compute the shared secret value \(P := [hd] Q_S\).

\textit{keyHandle} shall refer to a loaded ECC key. The sensitive portion of this key need not be loaded.

The curve parameters of the loaded ECC key are used to generate the ephemeral key.

\textbf{NOTE 1} This function is the equivalent of encrypting data to another object’s public key. The \textit{seed} value is used in a KDF to generate a symmetric key and that key is used to encrypt the data. Once the data is encrypted and the symmetric key discarded, only the object with the private portion of the \textit{keyHandle} will be able to decrypt it.

The \textit{zPoint} in the response may be encrypted using parameter encryption.
16.4.2 Command and Response

### Table 47 — TPM2_ECDH_KeyGen Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ECDH_KeyGen</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>keyHandle</td>
<td>Handle of a loaded ECC key public area.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
</tbody>
</table>

### Table 48 — TPM2_ECDH_KeyGen Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>zPoint</td>
<td>results of $P := h(d_e) Q_s$</td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>pubPoint</td>
<td>generated ephemeral public point ($Q_e$)</td>
</tr>
</tbody>
</table>
#include "InternalRoutines.h"
#include "ECDH_KeyGen_fp.h"
#ifdef TPM_ALG_ECC

Error Returns | Meaning
-------------|----------------------------------
TPM_RC_KEY | keyHandle does not reference a non-restricted decryption ECC key

TPM_RC
TPM2_ECDH_KeyGen(
    ECDH_KeyGen_In *in,       // IN: input parameter list
    ECDH_KeyGen_Out *out      // OUT: output parameter list
)
{
    OBJECT *eccKey;
    TPM2B_ECC_PARAMETER sensitive;
    TPM_RC result;

    // Input Validation
    eccKey = ObjectGet(in->keyHandle);
    if (eccKey->publicArea.type != TPM_ALG_ECC ||
        eccKey->publicArea.objectAttributes.restricted == SET ||
        eccKey->publicArea.objectAttributes.decrypt != SET)
        return TPM_RC_KEY + RC_ECDH_KeyGen_keyHandle;

    // Command Output
    do
    {
        // Create ephemeral ECC key
        CryptNewEccKey(eccKey->publicArea.parameters.eccDetail.curveID,
                       &out->pubPoint.t.point, &sensitive);

        out->pubPoint.t.size = TPMS_ECC_POINT_Marshal(&out->pubPoint.t.point,
                                                      NULL, NULL);

        // Compute Z
        result = CryptEccPointMultiply(&out->zPoint.t.point,
                                        eccKey->publicArea.parameters.eccDetail.curveID,
                                        &sensitive, eccKey->publicArea.unique.ecc);

        // The point in the key is not on the curve. Indicate that the key is bad.
        if (result == TPM_RC_ECC_POINT)
            return TPM_RC_KEY + RC_ECDH_KeyGen_keyHandle;

        // The other possible error is TPM_RC_NO_RESULT indicating that the
        // multiplication resulted in the point at infinity, so get a new
        // random key and start over (hardly ever happens).
        while (result != TPM_RC_SUCCESS);

        // Marshal the values to generate the point.
        out->zPoint.t.size = TPMS_ECC_POINT_Marshal(&out->zPoint.t.point, NULL, NULL);

        return TPM_RC_SUCCESS;
    }
    #endif
16.5 TPM2_ECDH_ZGen

16.5.1 General Description

This command uses the TPM to recover the $Z$ value from a public point ($Q_b$) and a private key ($d_s$). It will perform the multiplication of the provided $inPoint$ ($Q_b$) with the private key ($d_s$) and return the coordinates of the resultant point ($Z = (x_z, y_z) \equiv [h d_s]Q_b$; where $h$ is the cofactor of the curve).

$keyHandle$ shall refer to a loaded, ECC key (TPM_RC_KEY) with the restricted attribute CLEAR and the decrypt attribute SET (TPM_RC_ATTRIBUTES).

The scheme of the key referenced by $keyHandle$ is required to be either TPM_ALG_ECDH or TPM_ALG_NULL (TPM_RC_SCHEME).

$inPoint$ is required to be on the curve of the key referenced by $keyHandle$ (TPM_RC_ECC_POINT).

The parameters of the key referenced by $keyHandle$ are used to perform the point multiplication.
### 16.5.2 Command and Response

#### Table 49 — TPM2_ECDH_ZGen Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ECDH_ZGen</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@keyHandle</td>
<td>handle of a loaded ECC key</td>
</tr>
<tr>
<td></td>
<td>Auth Index: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Auth Role: USER</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>inPoint</td>
<td>a public key</td>
</tr>
</tbody>
</table>

#### Table 50 — TPM2_ECDH_ZGen Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
| TPM2B_ECC_POINT    | outPoint      | X and Y coordinates of the product of the multiplication \*

\[
Z = (x_Z, y_Z) = [hd_s]Q_B
\]
16.5.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ECDH_ZGen_fp.h"
#ifdef TPM_ALG_ECC

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>keyHandle does not reference a non-restricted decryption ECC key</td>
</tr>
<tr>
<td>TPM_RC_ECC_POINT</td>
<td>invalid argument</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>multiplying inPoint resulted in a point at infinity</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_ECDH_ZGen(
    ECDH_ZGen_In     *in,  // IN: input parameter list
    ECDH_ZGen_Out    *out   // OUT: output parameter list
)
{
    TPM_RC                   result;
    OBJECT                  *eccKey;

    // Input Validation
    eccKey = ObjectGet(in->keyHandle);

    // Input key must be a non-restricted, decrypt ECC key
    if(    eccKey->publicArea.type != TPM_ALG_ECC
         || eccKey->publicArea.objectAttributes.restricted == SET
         || eccKey->publicArea.objectAttributes.decrypt != SET
    )
        return TPM_RC_KEY + RC_ECDH_ZGen_keyHandle;

    // Command Output
    result = CryptEccPointMultiply(&out->outPoint.t.point,
        eccKey->publicArea.parameters.eccDetail.curveID,
        &eccKey->sensitive.sensitive.ecc,
        &in->inPoint.t.point);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_ECDH_ZGen_inPoint);

    out->outPoint.t.size = TPMS_ECC_POINT_Marshal(&out->outPoint.t.point,
        NULL, NULL);

    return TPM_RC_SUCCESS;
}
#endif
```
16.6 TPM2_ECC_Parameters

16.6.1 General Description

This command returns the parameters of an ECC curve identified by its TCG-assigned curveID.

16.6.2 Command and Response

Table 51 — TPM2_ECC_Parameters Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ECC_Parameters</td>
</tr>
<tr>
<td>TPM_ECC_CURVE</td>
<td>curveID</td>
<td>parameter set selector</td>
</tr>
</tbody>
</table>

Table 52 — TPM2_ECC_Parameters Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMS_ALGORITHM_DETAIL_ECC</td>
<td>parameters</td>
<td>ECC parameters for the selected curve</td>
</tr>
</tbody>
</table>
16.6.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ECC_Parameters_fp.h"
#ifndef TPM_ALG_ECC

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>Unsupported ECC curve ID</td>
</tr>
</tbody>
</table>

TPM_RC
TPM2_ECC_Parameters(
    ECC_Parameters_In *in,  // IN: input parameter list
    ECC_Parameters_Out *out  // OUT: output parameter list
) {
  // Command Output
  // Get ECC curve parameters
  if(CryptEccGetParameters(in->curveID, &out->parameters))
      return TPM_RC_SUCCESS;
  else
      return TPM_RC_VALUE + RC_ECC_Parameters_curveID;
}
#endif
```

16.7 TPM2_ZGen_2Phase

16.7.1 General Description

This command supports two-phase key exchange protocols. The command is used in combination with TPM2_EC_Ephemeral(). TPM2_EC_Ephemeral() generates an ephemeral key and returns the public point of that ephemeral key along with a numeric value that allows the TPM to regenerate the associated private key.

The input parameters for this command are a static public key (inQsU), an ephemeral key (inQeU) from party B, and the commitCounter returned by TPM2_EC_Ephemeral(). The TPM uses the counter value to regenerate the ephemeral private key (deV) and the associated public key (QeV). keyA provides the static ephemeral elements dsV and QsV. This provides the two pairs of ephemeral and static keys that are required for the schemes supported by this command.

The TPM will compute Z or Zs and Ze according to the selected scheme. If the scheme is not a two-phase key exchange scheme or if the scheme is not supported, the TPM will return TPM_RC_SCHEME.

It is an error if inQsB or inQeB are not on the curve of keyA (TPM_RC_ECC_POINT).

The two-phase key schemes that were assigned an algorithm ID as of the time of the publication of this specification are TPM_ALG_ECDH, TPM_ALG_ECMQV, and TPM_ALG_SM2.

If this command is supported, then support for TPM_ALG_ECDH is required. Support for TPM_ALG_ECMQV or TPM_ALG_SM2 is optional.

NOTE 1 If SM2 is supported and this command is supported, then the implementation is required to support the key exchange protocol of SM2, part 3.

For TPM_ALG_ECDH outZ1 will be Ze and outZ2 will Zs as defined in 6.1.1.2 of SP800-56A.
NOTE 2  A non-restricted decryption key using ECDH may be used in either TPM2_ECDH_ZGen() or TPM2_ZGen_2Phase as the computation done with the private part of keyA is the same in both cases.

For TPM_ALG_ECMQV or TPM_ALG_SM2 outZ1 will be Z and outZ2 will be an Empty Point.

NOTE 3  An Empty Point has two Empty Buffers as coordinates meaning the minimum size value for outZ2 will be four.

If the input scheme is TPM_ALG_ECDH, then outZ1 will be Z_s and outZ2 will be Z_e. For schemes like MQV (including SM2), outZ1 will contain the computed value and outZ2 will be an Empty Point.

NOTE  The Z values returned by the TPM are a full point and not just an x-coordinate.

If a computation of either Z produces the point at infinity, then the corresponding Z value will be an Empty Point.
## 16.7.2 Command and Response

### Table 53 — TPM2\_ZGen\_2Phase Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ZGen_2Phase</td>
</tr>
<tr>
<td>TPM_DH_OBJECT</td>
<td>@keyA</td>
<td>handle of an unrestricted decryption key ECC</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The private key referenced by this handle is used as $d_{S,A}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>inQsB</td>
<td>other party's static public key $(Q_{sB} = (X_{sB}, Y_{sB}))$</td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>inQeB</td>
<td>other party's ephemeral public key $Q_{eB} = (X_{eB}, Y_{eB})$</td>
</tr>
<tr>
<td>TPMI_ECC_KEY_EXCHANGE</td>
<td>inScheme</td>
<td>the key exchange scheme</td>
</tr>
<tr>
<td>UINT16</td>
<td>counter</td>
<td>value returned by TPM2_EC_Ephemeral()</td>
</tr>
</tbody>
</table>

### Table 54 — TPM2\_ZGen\_2Phase Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>outZ1</td>
<td>X and Y coordinates of the computed value (scheme dependent)</td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>outZ2</td>
<td>X and Y coordinates of the second computed value (scheme dependent)</td>
</tr>
</tbody>
</table>
16.7.3 Detailed Actions

This command uses the TPM to recover one or two Z values in a two phase key exchange protocol

```c
#include "InternalRoutines.h"
#include "ZGen_2Phase_fp.h"
#if defined TPM_ALG_ECC && (CC_ZGen_2Phase == YES)

TPM_RC ZGen_2Phase(
    ZGen_2Phase_In *in,     // IN: input parameter list
    ZGen_2Phase_Out *out    // OUT: output parameter list
)
{
    TPM_RC result;
    OBJECT *eccKey;
    TPM2B_ECC_PARAMETER r;
    TPM_ALG_ID scheme;

    // Input Validation
    eccKey = ObjectGet(in->keyA);
    if(eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RC_KEY + RC_ZGen_2Phase_keyA;
    // keyA must not be restricted and must be a decrypt key
    if( eccKey->publicArea.objectAttributes.restricted == SET
        || eccKey->publicArea.objectAttributes.decrypt != SET
    )
        return TPM_RC_ATTRIBUTES + RC_ZGen_2Phase_keyA;
    // if the scheme of keyA is TPM_ALG_NULL, then use the input scheme; otherwise
    // the input scheme must be the same as the scheme of keyA
    scheme = eccKey->publicArea.parameters.asymDetail.scheme.scheme;
    if(scheme != TPM_ALG_NULL)
    {
        if(scheme != in->inScheme)
            return TPM_RC_SCHEME + RC_ZGen_2Phase_inScheme;
    } else
    {
        scheme = in->inScheme;
        if(scheme == TPM_ALG_NULL)
            return TPM_RC_SCHEME + RC_ZGen_2Phase_inScheme;
    }

    // Input points must be on the curve of keyA
    if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,
        &in->inQsB.t.point))
        return TPM_RC_ECC_POINT + RC_ZGen_2Phase_inQsB;
    if(!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,
```
return TPM_RC_ECC_POINT + RC_ZGen_2Phase_inQeB;

if(!CryptGenerateR(&r, &in->counter,
    eccKey->publicArea.parameters.eccDetail.curveID,
    NULL))
    return TPM_RC_VALUE + RC_ZGen_2Phase_counter;

// Command Output

result = CryptEcc2PhaseKeyExchange(&out->outZ1.t.point,
    &out->outZ2.t.point,
    eccKey->publicArea.parameters.eccDetail.curveID,
    scheme,
    &eccKey->sensitive.sensitive.ecc,
    &r,
    &in->inQsB.t.point,
    &in->inQeB.t.point);

if(result != TPM_RC_SUCCESS)
    return result;

CryptEndCommit(in->counter);

return TPM_RC_SUCCESS;

#endif
17 Symmetric Primitives

17.1 Introduction

The commands in this clause provide low-level primitives for access to the symmetric algorithms implemented in the TPM that operate on blocks of data. These include symmetric encryption and decryption as well as hash and HMAC. All of the commands in this group are stateless. That is, they have no persistent state that is retained in the TPM when the command is complete.

For hashing, HMAC, and Events that require large blocks of data with retained state, the sequence commands are provided (see clause 1).

Some of the symmetric encryption/decryption modes use an IV. When an IV is used, it may be an initiation value or a chained value from a previous stage. The chaining for each mode is:
### Table 55 — Symmetric Chaining Process

<table>
<thead>
<tr>
<th>Mode</th>
<th>Chaining process</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPM_ALG_CTR</strong></td>
<td>The TPM will increment the entire IV provided by the caller. The last encrypted value will be returned to the caller as <code>ivOut</code>. This can be the input value to the next encrypted buffer. <code>ivIn</code> is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of <code>ivIn</code> is not correct, the TPM shall return TPM_RC_SIZE. <code>ivOut</code> will be the size of a cipher block and not the size of the last encrypted block. NOTE <code>ivOut</code> will be the value of the counter after the last block is encrypted. EXAMPLE 1 AES requires that <code>ivIn</code> be 128 bits (16 octets). EXAMPLE 2 If <code>ivIn</code> were 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 16 and four data blocks were encrypted, <code>ivOut</code> will have a value of 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04 16. All the bits of the IV are incremented as if it were an unsigned integer.</td>
</tr>
<tr>
<td><strong>TPM_ALG_OFB</strong></td>
<td>In Output Feedback (OFB), the output of the pseudo-random function (the block encryption algorithm) is XORed with a plaintext block to produce a ciphertext block. <code>ivOut</code> will be the value that was XORed with the last plaintext block. That value can be used as the <code>ivIn</code> for a next buffer. <code>ivIn</code> is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of <code>ivIn</code> is not correct, the TPM shall return TPM_RC_SIZE. <code>ivOut</code> will be the size of a cipher block and not the size of the last encrypted block.</td>
</tr>
<tr>
<td><strong>TPM_ALG_CBC</strong></td>
<td>For Cipher Block Chaining (CBC), a block of ciphertext is XORed with the next plaintext block and that block is encrypted. The encrypted block is then input to the encryption of the next block. The last ciphertext block then is used as an IV for the next buffer. Even though the last ciphertext block is evident in the encrypted data, it is also returned in <code>ivOut</code>. <code>ivIn</code> is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of <code>ivIn</code> is not correct, the TPM shall return TPM_RC_SIZE. inData is required to be an even multiple of the block encrypted by the selected algorithm and key combination. If the size of inData is not correct, the TPM shall return TPM_RC_SIZE.</td>
</tr>
<tr>
<td><strong>TPM_ALG_CFB</strong></td>
<td>Similar to CBC in that the last ciphertext block is an input to the encryption of the next block. <code>ivOut</code> will be the value that was XORed with the last plaintext block. That value can be used as the <code>ivIn</code> for a next buffer. <code>ivIn</code> is required to be the size of a block encrypted by the selected algorithm and key combination. If the size of <code>ivIn</code> is not correct, the TPM shall return TPM_RC_SIZE. <code>ivOut</code> will be the size of a cipher block and not the size of the last encrypted block.</td>
</tr>
<tr>
<td><strong>TPM_ALG_ECB</strong></td>
<td>Electronic Codebook (ECB) has no chaining. Each block of plaintext is encrypted using the key. ECB does not support chaining and <code>ivIn</code> shall be the Empty Buffer. <code>ivOut</code> will be the Empty Buffer. inData is required to be an even multiple of the block encrypted by the selected algorithm and key combination. If the size of inData is not correct, the TPM shall return TPM_RC_SIZE.</td>
</tr>
</tbody>
</table>
17.2 TPM2_EncryptDecrypt

17.2.1 General Description

This command performs symmetric encryption or decryption. 

keyHandle shall reference a symmetric cipher object (TPM_RC_KEY).

For a restricted key, mode shall be either the same as the mode of the key, or TPM_ALG_NULL (TPM_RC_VALUE). For an unrestricted key, mode may be the same or different from the mode of the key but both shall not be TPM_ALG_NULL (TPM_RC_VALUE).

If the TPM allows this command to be canceled before completion, then the TPM may produce incremental results and return TPM_RC_SUCCESS rather than TPM_RC_CANCEL. In such case, outData may be less than inData.
## 17.2.2 Command and Response

### Table 56 — TPM2_EncryptDecrypt Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_EncryptDecrypt</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@keyHandle</td>
<td>the symmetric key used for the operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_YES_NO</td>
<td>decrypt</td>
<td>if YES, then the operation is decryption; if NO, the operation is encryption</td>
</tr>
<tr>
<td>TPMI_ALG_SYM_MODE</td>
<td>mode</td>
<td>symmetric mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For a restricted key, this field shall match the default mode of the key or be TPM_ALG_NULL.</td>
</tr>
<tr>
<td>TPM2B_IV</td>
<td>ivIn</td>
<td>an initial value as required by the algorithm</td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>inData</td>
<td>the data to be encrypted/decrypted</td>
</tr>
</tbody>
</table>

### Table 57 — TPM2_EncryptDecrypt Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>outData</td>
<td>encrypted output</td>
</tr>
<tr>
<td>TPM2B_IV</td>
<td>ivOut</td>
<td>chaining value to use for IV in next round</td>
</tr>
</tbody>
</table>
17.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "EncryptDecrypt_fp.h"

<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

TPM2_EncryptDecrypt(
    EncryptDecrypt_In *in, // IN: input parameter list
    EncryptDecrypt_Out *out // OUT: output parameter list
) {
    OBJECT *symKey;
    UINT16 keySize;
    UINT16 blockSize;
    BYTE *key;
    TPM_ALG_ID alg;

    // Input Validation
    symKey = ObjectGet(in->keyHandle);

    // The input key should be a symmetric decrypt key.
    if (symKey->publicArea.type != TPM_ALG_SYMCIPHER
        || symKey->attributes.publicOnly == SET)
        return TPM_RC_KEY + RC_EncryptDecrypt_keyHandle;

    // If the input mode is TPM_ALG_NULL, use the key's mode
    if (in->mode == TPM_ALG_NULL)
        in->mode = symKey->publicArea.parameters.symDetail.sym.mode.sym;

    // If the key is restricted, the input sym mode should match the key's sym mode.
    if (symKey->publicArea.objectAttributes.restricted == SET
        && symKey->publicArea.parameters.symDetail.sym.mode.sym != in->mode)
        return TPM_RC_VALUE + RC_EncryptDecrypt_mode;

    // If the mode is null, then we have a problem.
    // Note: Construction of a TPMT_SYM_DEF does not allow the 'mode' to be TPM_ALG_NULL so setting in->mode to the mode of the key should have produced a valid mode. However, this is suspenderers.
    if (in->mode == TPM_ALG_NULL)
        return TPM_RC_VALUE + RC_EncryptDecrypt_mode;

    // The input iv for ECB mode should be null. All the other modes should have an iv size same as encryption block size
    blockSize = CryptGetSymmetricBlockSize(alg, keySize);
    if ((in->mode == TPM_ALG_ECB && in->ivIn.t.size != 0)
        || (in->mode != TPM_ALG_ECB && in->ivIn.t.size != blockSize))
        return TPM_RC_SIZE + RC_EncryptDecrypt_ivIn;
```

Error Returns | Meaning
---|---
TPM_RC_KEY | is not a symmetric decryption key with both public and private portions loaded
TPM_RC_SIZE | 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
TPM_RC_VALUE | keyHandle is restricted and the argument mode does not match the key's mode
// The input data size of CBC mode or ECB mode must be an even multiple of
// the symmetric algorithm's block size
if ((in->mode == TPM_ALG_CBC || in->mode == TPM_ALG_ECB)
    && (in->inData.t.size % blockSize) != 0)
    return TPM_RC_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 = in->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 = in->inData.t.size;
if(in->decrypt == YES)
{
    // Decrypt data to output
    CryptSymmetricDecrypt(out->outData.t.buffer,
        alg,
        keySize, in->mode, key,
        &(out->ivOut),
        in->inData.t.size,
        in->inData.t.buffer);
}
else
{
    // Encrypt data to output
    CryptSymmetricEncrypt(out->outData.t.buffer,
        alg,
        keySize,
        in->mode, key,
        &(out->ivOut),
        in->inData.t.size,
        in->inData.t.buffer);
}
return TPM_RC_SUCCESS;
17.3 TPM2_Hash

17.3.1 General Description

This command performs a hash operation on a data buffer and returns the results.

NOTE If the data buffer to be hashed is larger than will fit into the TPM's input buffer, then the sequence hash commands will need to be used.

If the results of the hash will be used in a signing operation that uses a restricted signing key, then the ticket returned by this command can indicate that the hash is safe to sign.

If the digest is not safe to sign, then the TPM will return a TPMT_TK_HASHCHECK with the hierarchy set to TPM_RH_NULL and digest set to the Empty Buffer.

If hierarchy is TPM_RH_NULL, then digest in the ticket will be the Empty Buffer.
### 17.3.2 Command and Response

**Table 58 — TPM2_Hash Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td>Shall have at least one session</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Hash</td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>data</td>
<td>data to be hashed</td>
</tr>
<tr>
<td>TPMI_ALG_HASH</td>
<td>hashAlg</td>
<td>algorithm for the hash being computed -- shall not be TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPMI_RH_HIERARCHY+</td>
<td>hierarchy</td>
<td>hierarchy to use for the ticket (TPM_RH_NULL allowed)</td>
</tr>
</tbody>
</table>

**Table 59 — TPM2_Hash Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>outHash</td>
<td>results</td>
</tr>
<tr>
<td>TPMT_TK_HASHCHECK</td>
<td>validation</td>
<td>ticket indicating that the sequence of octets used to compute outDigest did not start with TPM_GENERATED_VALUE will be a NULL ticket if the digest may not be signed with a restricted key</td>
</tr>
</tbody>
</table>
17.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Hash_fp.h"

TPM_RC TPM2_Hash(
    Hash_In *in, // IN: input parameter list
    Hash_Out *out // OUT: output parameter list
)
{

    HASH_STATE hashState;

    // Command Output
    // Output hash
    // Start hash stack
    out->outHash.t.size = CryptStartHash(in->hashAlg, &hashState);
    // Adding hash data
    CryptUpdateDigest2B(&hashState, &in->data.b);
    // Complete hash
    CryptCompleteHash2B(&hashState, &out->outHash.b);

    // Output ticket
    out->validation.tag = TPM_ST_HASHCHECK;
    out->validation.hierarchy = in->hierarchy;

    if (in->hierarchy == TPM_RH_NULL)
    {
        // Ticket is not required
        out->validation.hierarchy = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    else if (in->data.t.size >= sizeof(TPM_GENERATED)
            && !TicketIsSafe(&in->data.b))
    {
        // Ticket is not safe
        out->validation.hierarchy = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    }
    else
    {
        // Compute ticket
        TicketComputeHashCheck(in->hierarchy, &out->outHash, &out->validation);
    }

    return TPM_RC_SUCCESS;
}
```
17.4 TPM2_HMAC

17.4.1 General Description

This command performs an HMAC on the supplied data using the indicated hash algorithm. The caller shall provide proper authorization for use of handle.

If the sign attribute is not SET in the key referenced by handle then the TPM shall return TPM_RC_ATTRIBUTES. If the key type is not TPM_ALG_KEYEDHASH then the TPM shall return TPM_RC_TYPE.

If handle references a restricted key, then the hash algorithm specified in the key's scheme is used as the hash algorithm for the HMAC and the TPM shall return TPM_RC_VALUE if hashAlg is not TPM_ALG_NULL or the same algorithm as selected in the key's scheme.

NOTE 1 A restricted key may only have one of sign or decrypt SET and the default scheme may not be TPM_ALG_NULL. These restrictions are enforced by TPM2_Create() and TPM2_CreatePrimary().

If the key referenced by handle is not restricted, then the TPM will use hashAlg for the HMAC. However, if hashAlg is TPM_ALG_NULL the TPM will use the default scheme of the key.

If both hashAlg and the key default are TPM_ALG_NULL, the TPM shall return TPM_RC_VALUE.

NOTE A key may only have both sign and decrypt SET if the key is unrestricted. When both sign and decrypt are set, there is no default scheme for the key and the hash algorithm must be specified.
### 17.4.2 Command and Response

#### Table 60 — TPM2_HMAC Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_HMAC</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@handle</td>
<td>handle for the symmetric signing key providing the HMAC key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>buffer</td>
<td>HMAC data</td>
</tr>
<tr>
<td>TPMI_ALG_HASH+</td>
<td>hashAlg</td>
<td>algorithm to use for HMAC</td>
</tr>
</tbody>
</table>

#### Table 61 — TPM2_HMAC Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>outHMAC</td>
<td>the returned HMAC in a sized buffer</td>
</tr>
</tbody>
</table>
17.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "HMAC_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>key referenced by handle is not a signing key</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>key referenced by handle is not an HMAC key</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>hashAlg specified when the key is restricted is neither TPM_ALG_NULL</td>
</tr>
<tr>
<td></td>
<td>not equal to that of the key scheme; or both hashAlg and the key scheme's algorithm are TPM_ALG_NULL</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_HMAC(
    HMAC_In* in,         // IN: input parameter list
    HMAC_Out* out        // OUT: output parameter list
)
{
    HMAC_STATE hmacState;
    OBJECT *hmacObject;
    TPM_ALG_HASH hashAlg;
    TPMT_PUBLIC *publicArea;

    // Input Validation
    hmacObject = ObjectGet(in->handle);
    publicArea = &hmacObject->publicArea;

    // Make sure that the key is an HMAC signing key
    if (publicArea->type != TPM_ALG_KEYEDHASH)
        return TPM_RC_TYPE + RC_HMAC_handle;
    if (publicArea->objectAttributes.sign != SET)
        return TPM_RC_ATTRIBUTES + RC_HMAC_handle;

    // Assume that the key default scheme is used
    hashAlg = publicArea->parameters.keyedHashDetail.scheme.details.hmac.hashAlg;

    // if the key is restricted, then need to use the scheme of the key and the
    // input algorithm must be TPM_ALG_NULL or the same as the key scheme
    if (publicArea->objectAttributes.restricted == SET)
        {  
            if ((in->hashAlg != TPM_ALG_NULL && in->hashAlg != hashAlg)
                hashAlg = TPM_ALG_NULL;
            }
    else
        {  
            // for a non-restricted key, use hashAlg if it is provided;
            if ((in->hashAlg != TPM_ALG_NULL)
                hashAlg = in->hashAlg;
            }

    // if the hashAlg is TPM_ALG_NULL, then the input hashAlg is not compatible
    // with the key scheme or type
    if (hashAlg == TPM_ALG_NULL)
        return TPM_RC_VALUE + RC_HMAC_hashAlg;

    // Command Output
```

```
50   // Start HMAC stack
51   out->outHMAC.t.size = CryptStartHMAC2B(hashAlg,
52       &hmacObject->sensitive.sensitive.bits.b,
53       &hmacState);
54   // Adding HMAC data
55   CryptUpdateDigest2B(&hmacState, &in->buffer.b);
56   // Complete HMAC
57   CryptCompleteHMAC2B(&hmacState, &out->outHMAC.b);
58   return TPM_RC_SUCCESS;
59 }
60 }
18 Random Number Generator

18.1 TPM2_GetRandom

18.1.1 General Description

This command returns the next bytesRequested octets from the random number generator (RNG).

NOTE 1 It is recommended that a TPM implement the RNG in a manner that would allow it to return RNG octets such that the frequency of bytesRequested being more than the number of octets available is an infrequent occurrence.

If bytesRequested is more than will fit into a TPM2B_DIGEST on the TPM, no error is returned but the TPM will only return as much data as will fit into a TPM2B_DIGEST buffer for the TPM.

NOTE 2 TPM2B_DIGEST is large enough to hold the largest digest that may be produced by the TPM. Because that digest size changes according to the implemented hashes, the maximum amount of data returned by this command is TPM implementation-dependent.
18.1.2 Command and Response

Table 62 — TPM2_GetRandom Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_GetRandom</td>
</tr>
<tr>
<td>UINT16</td>
<td>bytesRequested</td>
<td>number of octets to return</td>
</tr>
</tbody>
</table>

Table 63 — TPM2_GetRandom Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>randomBytes</td>
<td>the random octets</td>
</tr>
</tbody>
</table>
18.1.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "GetRandom_fp.h"

TPM_RC
TPM2_GetRandom(
    GetRandom_In *in, // IN: input parameter list
    GetRandom_Out *out // OUT: output parameter list
)
{
    // Command Output

    // if the requested bytes exceed the output buffer size, generates the
    // maximum bytes that the output buffer allows
    if (in->bytesRequested > sizeof(TPMU_HA))
        out->randomBytes.t.size = sizeof(TPMU_HA);
    else
        out->randomBytes.t.size = in->bytesRequested;

    CryptGenerateRandom(out->randomBytes.t.size, out->randomBytes.t.buffer);

    return TPM_RC_SUCCESS;
}
```
18.2 TPM2_StirRandom

18.2.1 General Description

This command is used to add "additional information" to the RNG state.

NOTE The "additional information" is as defined in SP800-90A.

The inData parameter may not be larger than 128 octets.
18.2.2 Command and Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_StirRandom (NV)</td>
</tr>
<tr>
<td>TPM2B_SENSITIVE_DATA</td>
<td>inData</td>
<td>additional information</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
18.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "StirRandom_fp.h"
TPM_RC
TPM2_StirRandom(
    StirRandom_In      *in     // IN: input parameter list
) {
    // Internal Data Update
    CryptStirRandom(in->inData.t.size, in->inData.t.buffer);
    return TPM_RC_SUCCESS;
}
```
19  Hash/HMAC/Event Sequences

19.1  Introduction

All of the commands in this group are to support sequences for which an intermediate state must be maintained. For a description of sequences, see “Hash, HMAC, and Event Sequences” in Part 1.

19.2  TPM2_HMAC_Start

19.2.1  General Description

This command starts an HMAC sequence. The TPM will create and initialize an HMAC sequence structure, assign a handle to the sequence, and set the authValue of the sequence object to the value in auth.

NOTE 1  The structure of a sequence object is vendor-dependent.

The caller shall provide proper authorization for use of handle.

If the sign attribute is not SET in the key referenced by handle then the TPM shall return TPM_RC_ATTRIBUTES. If the key type is not TPM_ALG_KEYEDHASH then the TPM shall return TPM_RC_TYPE.

If handle references a restricted key, then the hash algorithm specified in the key's scheme is used as the hash algorithm for the HMAC and the TPM shall return TPM_RC_VALUE if hashAlg is not TPM_ALG_NULL or the same algorithm in the key's scheme.

If the key referenced by handle is not restricted, then the TPM will use hashAlg for the HMAC; unless hashAlg is TPM_ALG_NULL in which case it will use the default scheme of the key.

### Table 66 — Hash Selection Matrix

<table>
<thead>
<tr>
<th>handle→restricted (key's restricted attribute)</th>
<th>handle→scheme (hash algorithm from key's scheme)</th>
<th>hashAlg</th>
<th>hash used</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLEAR (unrestricted)</td>
<td>TPM_ALG_NULL(1)</td>
<td>TPM_ALG_NULL</td>
<td>error(2) (TPM_RC_SCHEME)</td>
</tr>
<tr>
<td>CLEAR</td>
<td>don’t care</td>
<td>valid hash</td>
<td>hashAlg</td>
</tr>
<tr>
<td>CLEAR</td>
<td>valid hash</td>
<td>TPM_ALG_NULL</td>
<td>handle→scheme</td>
</tr>
<tr>
<td>SET (restricted)</td>
<td>valid hash(3)</td>
<td>TPM_ALG_NULL</td>
<td>handle→scheme</td>
</tr>
<tr>
<td>SET</td>
<td>valid hash(3)</td>
<td>same as handle→scheme</td>
<td>handle→scheme</td>
</tr>
<tr>
<td>SET</td>
<td>valid hash(3)</td>
<td>not same as handle→scheme</td>
<td>error(4) (TPM_RC_SCHEME)</td>
</tr>
</tbody>
</table>

NOTES:

1) The scheme for the handle may only be TPM_ALG_NULL if both sign and decrypt are SET.
2) A hash algorithm is required for the HMAC.
3) A restricted key is required to have a scheme with a valid hash algorithm. A restricted key may not have both sign and decrypt SET.
4) The scheme for a restricted key cannot be overridden.
### 19.2.2 Command and Response

Table 67 — TPM2_HMAC_Start Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_HMAC_Start handle of an HMAC key</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@handle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_AUTH</td>
<td>auth</td>
<td>authorization value for subsequent use of the sequence</td>
</tr>
<tr>
<td>TPMI_ALG_HASH+</td>
<td>hashAlg</td>
<td>the hash algorithm to use for the HMAC</td>
</tr>
</tbody>
</table>

Table 68 — TPM2_HMAC_Start Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>sequenceHandle</td>
<td>a handle to reference the sequence</td>
</tr>
</tbody>
</table>
19.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "HMAC_Start_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>key referenced by handle is not a signing key</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>no space to create an internal object</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>key referenced by handle is not an HMAC key</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>hashAlg specified when the key is restricted is neither TPM_ALG_NULL not equal to that of the key scheme; or both hashAlg and the key scheme's algorithm are TPM_ALG_NULL</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_HMAC_Start(
  HMAC_Start_In       *in,     // IN: input parameter list
  HMAC_Start_Out      *out    // OUT: output parameter list
)
{
  OBJECT                  *hmacObject;
  TPMT_PUBLIC             *publicArea;
  TPM_ALG_ID               hashAlg;

  // Input Validation

  // Get HMAC key object and public area pointers
  hmacObject = ObjectGet(in->handle);
  publicArea = &hmacObject->publicArea;

  // Make sure that the key is an HMAC signing key
  if(publicArea->type != TPM_ALG_KEYEDHASH)
    return TPM_RC_TYPE + RC_HMAC_Start_handle;
  if(publicArea->objectAttributes.sign != SET)
    return TPM_RC_ATTRIBUTES + RC_HMAC_Start_handle;

  // Assume that the key default scheme is used
  hashAlg = publicArea->parameters.keyedHashDetail.scheme.details.hmac.hashAlg;

  // if the key is restricted, then need to use the scheme of the key and the
  // input algorithm must be TPM_ALG_NULL or the same as the key scheme
  if(publicArea->objectAttributes.restricted == SET)
  {
    if((in->hashAlg != TPM_ALG_NULL) && in->hashAlg != hashAlg)
      hashAlg = TPM_ALG_NULL;
  }
  else
  {
    // for a non-restricted key, use hashAlg if it is provided;
    if(in->hashAlg != TPM_ALG_NULL)
      hashAlg = in->hashAlg;
  }

  // if the algorithm selection ended up with TPM_ALG_NULL, then either the
  // schemes are not compatible or no hash was provided and both conditions
  // are errors.
  if(hashAlg == TPM_ALG_NULL)
    return TPM_RC_VALUE + RC_HMAC_Start_hashAlg;

  // Internal Data Update
```
48   // Create a HMAC sequence object. A TPM_RC_OBJECT_MEMORY error may be
49   // returned at this point
50   return ObjectCreateHMACSequence(hashAlg,
51                        in->handle,
52                        &in->auth,
53                        &out->sequenceHandle);
19.3 TPM2_HashSequenceStart

19.3.1 General Description

This command starts a hash or an Event sequence. If hashAlg is an implemented hash, then a hash sequence is started. If hashAlg is TPM_ALG_NULL, then an Event sequence is started. If hashAlg is neither an implemented algorithm nor TPM_ALG_NULL, then the TPM shall return TPM_RC_HASH.

Depending on hashAlg, the TPM will create and initialize a hash sequence structure or an Event sequence structure. Additionally, it will assign a handle to the sequence and set the authValue of the sequence to the value in auth. A sequence structure for an Event (hashAlg = TPM_ALG_NULL) contains a hash context for each of the PCR banks implemented on the TPM.
19.3.2 Command and Response

Table 69 — TPM2_HashSequenceStart Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_HashSequenceStart</td>
</tr>
<tr>
<td>TPM2B_AUTH</td>
<td>auth</td>
<td>authorization value for subsequent use of the sequence</td>
</tr>
<tr>
<td>TPMI_ALG_HASH+</td>
<td>hashAlg</td>
<td>the hash algorithm to use for the hash sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>An Event sequence starts if this is TPM_ALG_NULL.</td>
</tr>
</tbody>
</table>

Table 70 — TPM2_HashSequenceStart Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>sequenceHandle</td>
<td></td>
</tr>
</tbody>
</table>
19.3.3 Detailed Actions

#include "InternalRoutines.h"
#include "HashSequenceStart_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>no space to create an internal object</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_HashSequenceStart(
  HashSequenceStart_In *in,      // IN: input parameter list
  HashSequenceStart_Out *out    // OUT: output parameter list
)
{
  // Internal Data Update
  if (in->hashAlg == TPM_ALG_NULL)
    // Start a event sequence. A TPM_RC_OBJECT_MEMORY error may be
    // returned at this point
    return ObjectCreateEventSequence(&in->auth, &out->sequenceHandle);

  // Start a hash sequence. A TPM_RC_OBJECT_MEMORY error may be
  // returned at this point
  return ObjectCreateHashSequence(in->hashAlg, &in->auth, &out->sequenceHandle);
}
19.4 TPM2_SequenceUpdate

19.4.1 General Description

This command is used to add data to a hash or HMAC sequence. The amount of data in buffer may be any size up to the limits of the TPM.

NOTE In all TPM, a buffer size of 1,024 octets is allowed.

Proper authorization for the sequence object associated with sequenceHandle is required. If an authorization or audit of this command requires computation of a cpHash and an rpHash, the Name associated with sequenceHandle will be the Empty Buffer.

If the command does not return TPM_RC_SUCCESS, the state of the sequence is unmodified.

If the sequence is intended to produce a digest that will be signed by a restricted signing key, then the first block of data shall contain sizeof(TPM_GENERATED) octets and the first octets shall not be TPM_GENERATED_VALUE.

NOTE This requirement allows the TPM to validate that the first block is safe to sign without having to accumulate octets over multiple calls.
### 19.4.2 Command and Response

#### Table 71 — TPM2_SequenceUpdate Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_SequenceUpdate</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@sequenceHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>buffer</td>
<td>data to be added to hash</td>
</tr>
</tbody>
</table>

#### Table 72 — TPM2_SequenceUpdate Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
19.4.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "SequenceUpdate_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_MODE</td>
<td>sequenceHandle does not reference a hash or HMAC sequence object</td>
</tr>
</tbody>
</table>

TPM_RC
TPM2_SequenceUpdate(
    SequenceUpdate_In *in  // IN: input parameter list
    )
{
    OBJECT *object;

    // Input Validation
    object = ObjectGet(in->sequenceHandle);

    // Check that referenced object is a sequence object.
    if(!ObjectIsSequence(object))
        return TPM_RC_MODE + RC_SequenceUpdate_sequenceHandle;

    // Internal Data Update
    if(object->attributes.eventSeq == SET)
        { // Update event sequence object
        UINT32 i;
        HASH_OBJECT *hashObject = (HASH_OBJECT *)object;
        for(i = 0; i < HASH_COUNT; i++)
        {
            // Update sequence object
            CryptUpdateDigest2B(&hashObject->state.hashState[i], &in->buffer.b);
        }
    }
    else
        { // Update hash/HMAC sequence object
        HASH_OBJECT *hashObject = (HASH_OBJECT *)object;

        // Is this the first block of the sequence
        if(hashObject->attributes.hashSeq == SET)
            { // If so, indicate that first block was received
            hashObject->attributes.firstBlock = SET;

            // Check the first block to see if the first block can contain
            // the TPM_GENERATED_VALUE. If it does, it is not safe for
            // a ticket.
            if(TicketIsSafe(&in->buffer.b))
                hashObject->attributes.ticketSafe = SET;
            }

            // Update sequence object hash/HMAC stack
            CryptUpdateDigest2B(&hashObject->state.hashState[0], &in->buffer.b);
        }
```
} else if (object->attributes.hmacSeq == SET) {
    HASH_OBJECT *hashObject = (HASH_OBJECT *)object;
    // Update sequence object hash/HMAC stack
    CryptUpdateDigest2B(hashObject->state.hmacState, &in->buffer.b);
} return TPM_RC_SUCCESS;
19.5 TPM2_SequenceComplete

19.5.1 General Description

This command adds the last part of data, if any, to a hash/HMAC sequence and returns the result.

NOTE 1 This command is not used to complete an Event sequence. TPM2_EventSequenceComplete() is used for that purpose.

For a hash sequence, if the results of the hash will be used in a signing operation that uses a restricted signing key, then the ticket returned by this command can indicate that the hash is safe to sign.

If the digest is not safe to sign, then validation will be a TPMT_TK_HASHCHECK with the hierarchy set to TPM_RH_NULL and digest set to the Empty Buffer.

NOTE 2 Regardless of the contents of the first octets of the hashed message, if the first buffer sent to the TPM had fewer than sizeof(TPM_GENERATED) octets, then the TPM will operate as if digest is not safe to sign.

If sequenceHandle references an Event sequence, then the TPM shall return TPM_RC_MODE.

Proper authorization for the sequence object associated with sequenceHandle is required. If an authorization or audit of this command requires computation of a cpHash and an rpHash, the Name associated with sequenceHandle will be the Empty Buffer.

If this command completes successfully, the sequenceHandle object will be flushed.
19.5.2 Command and Response

<table>
<thead>
<tr>
<th>Table 73 — TPM2_SequenceComplete Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_CC</td>
</tr>
<tr>
<td>TPM_DH_OBJECT</td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
</tr>
<tr>
<td>TPM_RH_HIERARCHY+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 74 — TPM2_SequenceComplete Response</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TPM_ST</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_RC</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
</tr>
<tr>
<td>TPMT_TK_HASHCHECK</td>
</tr>
</tbody>
</table>
19.5.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "SequenceComplete_fp.h"
#include <Platform.h>

Error Returns | Meaning
--- | ---
TPM_RC_TYPE | `sequenceHandle` does not reference a hash or HMAC sequence object

TPM_RC

TPM2_SequenceComplete(
    SequenceComplete_In *in, // IN: input parameter list
    SequenceComplete_Out *out // OUT: output parameter list
)
{
    OBJECT *object;

    // Input validation
    object = ObjectGet(in->sequenceHandle);
    // input handle must be a hash or HMAC sequence object.
    if (object->attributes.hashSeq == CLEAR
        && object->attributes.hmacSeq == CLEAR)
        return TPM_RC_MODE + RC_SequenceComplete_sequenceHandle;

    // Command Output
    if(object->attributes.hashSeq == SET) // sequence object for hash
    {
        // Update last piece of data
        HASH_OBJECT *hashObject = (HASH_OBJECT *)object;
        CryptUpdateDigest2B(&hashObject->state.hashState[0], &in->buffer.b);

        // Complete hash
        out->result.t.size = CryptGetHashDigestSize(
            CryptGetContextAlg(&hashObject->state.hashState[0]));
        CryptCompleteHash2B(&hashObject->state.hashState[0], &out->result.b);

        // Check if the first block of the sequence has been received
        if(hashObject->attributes.firstBlock == CLEAR)
        {
            // If not, then this is the first block so see if it is 'safe'
            // to sign.
            if(TicketIsSafe(&in->buffer.b))
                hashObject->attributes.ticketSafe = SET;
        }

        // Output ticket
        out->validation.tag = TPM_ST_HASHCHECK;
        out->validation.hierarchy = in->hierarchy;
        if(in->hierarchy == TPM_RH_NULL)
        {
            // Ticket is not required
            out->validation.digest.t.size = 0;
        }
    }
```

54     }  
55 else if (object-&gt;attributes.ticketSafe == CLEAR)  
56     {  
57         // Ticket is not safe to generate  
58         out-&gt;validation.hierarchy = TPM_RH_NULL;  
59         out-&gt;validation.digest.t.size = 0;  
60     }  
61 else  
62     {  
63         // Compute ticket  
64         TicketComputeHashCheck(out-&gt;validation.hierarchy,  
65             &amp;out-&gt;result, &amp;out-&gt;validation);  
66     }  
67 else  
68     {  
69         HASH_OBJECT     *hashObject = (HASH_OBJECT *)object;  
70         // Update last piece of data  
71         CryptUpdateDigest2B(&amp;hashObject-&gt;state.hmacState, &amp;in-&gt;buffer.b);  
72         // Complete hash/HMAC  
73         out-&gt;result.t.size =  
74             CryptGetHashDigestSize(  
75                 CryptGetContextAlg(&amp;hashObject-&gt;state.hmacState.hashState));  
76             CryptCompleteHMAC2B(&amp;(hashObject-&gt;state.hmacState), &amp;out-&gt;result.b);  
77         // No ticket is generated for HMAC sequence  
78         out-&gt;validation.tag = TPM_ST_HASHCHECK;  
79         out-&gt;validation.hierarchy = TPM_RH_NULL;  
80         out-&gt;validation.digest.t.size = 0;  
81     }  
82  
83 // Internal Data Update  
84  
85 // mark sequence object as evict so it will be flushed on the way out  
86 object-&gt;attributes.evict = SET;  
87  
88 return TPM_RC_SUCCESS;  
89 }
19.6 TPM2_EventSequenceComplete

19.6.1 General Description

This command adds the last part of data, if any, to an Event sequence and returns the result in a digest list. If `pcrHandle` references a PCR and not TPM_RH_NULL, then the returned digest list is processed in the same manner as the digest list input parameter to TPM2_PCR_Extend() with the `pcrHandle` in each bank extended with the associated digest value.

If `sequenceHandle` references a hash or HMAC sequence, the TPM shall return TPM_RC_MODE.

Proper authorization for the sequence object associated with `sequenceHandle` is required. If an authorization or audit of this command requires computation of a `cpHash` and an `rpHash`, the Name associated with `sequenceHandle` will be the Empty Buffer.

If this command completes successfully, the `sequenceHandle` object will be flushed.
19.6.2 Command and Response

Table 75 — TPM2_EventSequenceComplete Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_EventSequenceComplete (NV F)</td>
</tr>
<tr>
<td>TPMI_DH_PCR+</td>
<td>@ pcrHandle</td>
<td>PCR to be extended with the Event data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@sequenceHandle</td>
<td>authorization for the sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>buffer</td>
<td>data to be added to the Event</td>
</tr>
</tbody>
</table>

Table 76 — TPM2_EventSequenceComplete Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPMI_CC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPML_DIGEST_VALUES</td>
<td>results</td>
<td>list of digests computed for the PCR</td>
</tr>
</tbody>
</table>
19.6.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "EventSequenceComplete_fp.h"

Error Returns | Meaning
--------------|-------------------
TPM_RC_LOCALITY | PCR extension is not allowed at the current locality
TPM_RC_MODE | input handle is not a valid event sequence object

TPM_RC

TPM2_EventSequenceComplete(
    EventSequenceComplete_In *in,  // IN: input parameter list
    EventSequenceComplete_Out *out  // OUT: output parameter list
)
{
    TPM_RC result;
    HASH_OBJECT *hashObject;
    UINT32 i;
    TPM_ALG_ID hashAlg;

    // Input validation
    // get the event sequence object pointer
    hashObject = (HASH_OBJECT *)ObjectGet(in->sequenceHandle);

    // input handle must reference an event sequence object
    if(hashObject->attributes.eventSeq != SET)
    {
        return TPM_RC_MODE + RC_EventSequenceComplete_sequenceHandle;
    }

    // see if a PCR extend is requested in call
    if(in->pcrHandle != TPM_RH_NULL)
    {
        // see if extend of the PCR is allowed at the locality of the command,
        if(!PCRIsExtendAllowed(in->pcrHandle))
        {
            return TPM_RC_LOCALITY;
        }

        // if an extend is going to take place, then check to see if there has
        // been an orderly shutdown. If so, and the selected PCR is one of the
        // state saved PCR, then the orderly state has to change. The orderly state
        // does not change for PCR that are not preserved.
        // NOTE: This doesn't just check for Shutdown(STATE) because the orderly
        // state will have to change if this is a state-saved PCR regardless
        // of the current state. This is because a subsequent Shutdown(STATE) will
        // check to see if there was an orderly shutdown and not do anything if
        // there was. So, this must indicate that a future Shutdown(STATE) has
        // something to do.
        if(gp.orderlyState != SHUTDOWN_NONE && PCRIsStateSaved(in->pcrHandle))
        {
            result = NvIsAvailable();
            if(result != TPM_RC_SUCCESS) return result;
            g_clearOrderly = TRUE;
        }
    }

    // Command Output
    out->results.count = 0;
    for(i = 0; i < HASH_COUNT; i++)
    {
        hashAlg = CryptGetHashAlgByIndex(i);
    }
```

// Update last piece of data
CryptUpdateDigest2B(&hashObject->state.hashState[i], &in->buffer.b);

// Complete hash
out->results.digests[out->results.count].hashAlg = hashAlg;
CryptCompleteHash(&hashObject->state.hashState[i],
  CryptGetHashDigestSize(hashAlg),
  (BYTE *)&out->results.digests[out->results.count].digest);

// Extend PCR
if (in->pcrHandle != TPM_RH_NULL)
  PCRExtend(in->pcrHandle, hashAlg,
    CryptGetHashDigestSize(hashAlg),
    (BYTE *)&out->results.digests[out->results.count].digest);
out->results.count++;

// Internal Data Update
// mark sequence object as evict so it will be flushed on the way out
hashObject->attributes.evict = SET;
return TPM_RC_SUCCESS;

20 Attestation Commands

20.1 Introduction

The attestation commands cause the TPM to sign an internally generated data structure. The contents of the data structure vary according to the command.

For all signing commands, provisions are made for the caller to provide a scheme to be used for the signing operation. This scheme will be applied only if the scheme of the key is TPM_ALG_NULL. If the scheme for signHandle is not TPM_ALG_NULL, then inScheme.scheme shall be TPM_ALG_NULL or the same as scheme in the public area of the key. If the scheme for signHandle is TPM_ALG_NULL, then inScheme will be used for the signing operation and may not be TPM_ALG_NULL. The TPM shall return TPM_RC_SCHEME to indicate that the scheme is not appropriate.

For a signing key that is not restricted, the caller may specify the scheme to be used as long as the scheme is compatible with the family of the key (for example, TPM_ALG_RSAPSS cannot be selected for an ECC key). If the caller sets scheme to TPM_ALG_NULL, then the default scheme of the key is used.

If the handle for the signing key (signHandle) is TPM_RH_NULL, then all of the actions of the command are performed and the attestation block is "signed" with the NULL Signature.

NOTE 1 This mechanism is provided so that additional commands are not required to access the data that might be in an attestation structure.

NOTE 2 When signHandle is TPM_RH_NULL, scheme is still required to be a valid signing scheme (may be TPM_ALG_NULL), but the scheme will have no effect on the format of the signature. It will always be the NULL Signature.

TPM2_NV_Certify() is an attestation command that is documented in 1. The remaining attestation commands are collected in the remainder of this clause.

Each of the attestation structures contains a TPMS_CLOCK_INFO structure and a firmware version number. These values may be considered privacy-sensitive, because they would aid in the correlation of attestations by different keys. To provide improved privacy, the resetCount, restartCount, and firmwareVersion numbers are obfuscated when the signing key is not in the Endorsement or Platform hierarchies.

The obfuscation value is computed by:

\[
\text{obfuscation} := \text{KDFa}(\text{signHandle} \rightarrow \text{nameAlg}, \text{shProof}, \text{"OBFSUCATE"}, \text{signHandle} \rightarrow \text{QN}, 0, 128)
\] (3)

Of the returned 128 bits, 64 bits are added to the versionNumber field of the attestation structure; 32 bits are added to the clockInfo.resetCount and 32 bits are added to the clockInfo.restartCount. The order in which the bits are added is implementation-dependent.

NOTE 3 The obfuscation value for each signing key will be unique to that key in a specific location. That is, each version of a duplicated signing key will have a different obfuscation value.

When the signing key is TPM_RH_NULL, the data structure is produced but not signed; and the values in the signed data structure are obfuscated. When computing the obfuscation value for TPM_RH_NULL, the hash used for context integrity is used.

NOTE 4 The QN for TPM_RH_NULL is TPM_RH_NULL.

If the signing scheme of signHandle is an anonymous scheme, then the attestation blocks will not contain the Qualified Name of the signHandle.

Each of the attestation structures allows the caller to provide some qualifying data (qualifyingData). For most signing schemes, this value will be placed in the TPMS_ATTEST.extraData parameter that is then
hashed and signed. However, for some schemes such as ECDAA, the qualifyingData is used in a different manner (for details, see “ECDAA” in Part 1).
20.2  TPM2_Certify

20.2.1  General Description

The purpose of this command is to prove that an object with a specific Name is loaded in the TPM. By certifying that the object is loaded, the TPM warrants that a public area with a given Name is self-consistent and associated with a valid sensitive area. If a relying party has a public area that has the same Name as a Name certified with this command, then the values in that public area are correct.

NOTE 1  See 20.1 for description of how the signing scheme is selected.

Authorization for objectHandle requires ADMIN role authorization. If performed with a policy session, the session shall have a policySession→commandCode set to TPM_CC_Certify. This indicates that the policy that is being used is a policy that is for certification, and not a policy that would approve another use. That is, authority to use an object does not grant authority to certify the object.

The object may be any object that is loaded with TPM2_Load() or TPM2_CreatePrimary(). An object that only has its public area loaded cannot be certified.

NOTE 2  The restriction occurs because the Name is used to identify the object being certified. If the TPM has not validated that the public area is associated with a matched sensitive area, then the public area may not represent a valid object and cannot be certified.

The certification includes the Name and Qualified Name of the certified object as well as the Name and the Qualified Name of the certifying object.
## 20.2.2 Command and Response

### Table 77 — TPM2_Certify Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Certify handle of the object to be certified</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@objectHandle</td>
<td>Auth Index: 1, Auth Role: ADMIN</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>@signHandle</td>
<td>handle of the key used to sign the attestation structure</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>qualifyingData</td>
<td>user provided qualifying data</td>
</tr>
<tr>
<td>TPMT_SIG_SCHEME+</td>
<td>inScheme</td>
<td>signing scheme to use if the scheme for signHandle is TPM_ALG_NULL</td>
</tr>
</tbody>
</table>

### Table 78 — TPM2_Certify Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ATTEST</td>
<td>certifyInfo</td>
<td>the structure that was signed</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>signature</td>
<td>the asymmetric signature over certifyInfo using the key referenced by signHandle</td>
</tr>
</tbody>
</table>
20.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "Certify_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>key referenced by signHandle is not a signing key</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>inScheme is not compatible with signHandle</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>digest generated for inScheme is greater or has larger size than the modulus of signHandle, or the buffer for the result in signature is too small (for an RSA key); invalid commit status (for an ECC key with a split scheme).</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_Certify(
    Certify_In *in,       // IN: input parameter list
    Certify_Out *out      // OUT: output parameter list
)
{
    TPM_RC                  result;
    TPMS_ATTEST             certifyInfo;

    // Command Output
    // Filling in attest information
    // Common fields
    result = FillInAttestInfo(in->signHandle,
                               &in->inScheme,
                               &in->qualifyingData,
                               &certifyInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_Certify_signHandle;
        else
            return RcSafeAddToResult(result, RC_Certify_inScheme);
    }
    // Certify specific fields
    // Attestation type
    certifyInfo.type = TPM_ST_ATTEST_CERTIFY;
    // Certified object name
    certifyInfo.attested.certify.name.t.size =
        ObjectGetName(in->objectHandle,
                      &certifyInfo.attested.certify.name.t.name);
    // Certified object qualified name
    ObjectGetQualifiedName(in->objectHandle,
                            &certifyInfo.attested.certify.qualifiedName);

    // Sign attestation structure.  A NULL signature will be returned if
    // signHandle is TPM_RH_NULL.  A TPM_RC_NV UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned
    // by SignAttestInfo()
    result = SignAttestInfo(in->signHandle,
                            &in->inScheme,
                            &certifyInfo,
                            &in->qualifyingData,
                            &out->certifyInfo,
```
49       $out->signature);
50
51     // TPM_RC_ATTRIBUTES cannot be returned here as FillInAttestInfo would already
52     // have returned TPM_RC_KEY
53     pAssert(result != TPM_RC_ATTRIBUTES);
54
55     if (result != TPM_RC_SUCCESS)
56         return result;
57
58     // orderly state should be cleared because of the reporting of clock info
59     // if signing happens
60     if (in->signHandle != TPM_RH_NULL)
61         g_clearOrderly = TRUE;
62
63     return TPM_RC_SUCCESS;
20.3 TPM2_CertifyCreation

20.3.1 General Description

This command is used to prove the association between an object and its creation data. The TPM will validate that the ticket was produced by the TPM and that the ticket validates the association between a loaded public area and the provided hash of the creation data (creationHash).

NOTE 1 See 20.1 for description of how the signing scheme is selected.

The TPM will create a test ticket using the Name associated with objectHandle and creationHash as:

$$\text{HMAC}(\text{proof}, (\text{TPM_ST_CREATION || objectHandle} \rightarrow \text{Name || creationHash}))$$ (4)

This ticket is then compared to creation ticket. If the tickets are not the same, the TPM shall return TPM_RC_TICKET.

If the ticket is valid, then the TPM will create a TPMS_ATTEST structure and place creationHash of the command in the creationHash field of the structure. The Name associated with objectHandle will be included in the attestation data that is then signed using the key associated with signHandle.

NOTE 2 If signHandle is TPM_RH_NULL, the TPMS_ATTEST structure is returned and signature is a NULL Signature.

ObjectHandle may be any object that is loaded with TPM2_Load() or TPM2_CreatePrimary().
20.3.2 Command and Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_CertifyCreation</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>@signHandle</td>
<td>handle of the key that will sign the attestation block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>objectHandle</td>
<td>the object associated with the creation data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>qualifyingData</td>
<td>user-provided qualifying data</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>creationHash</td>
<td>hash of the creation data produced by TPM2_Create() or TPM2_CreatePrimary()</td>
</tr>
<tr>
<td>TPMT_SIG_SCHEME+</td>
<td>inScheme</td>
<td>signing scheme to use if the scheme for signHandle is TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPMT_TK_CREATION</td>
<td>creationTicket</td>
<td>ticket produced by TPM2_Create() or TPM2_CreatePrimary()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ATTEST</td>
<td>certifyInfo</td>
<td>the structure that was signed</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>signature</td>
<td>the signature over certifyInfo</td>
</tr>
</tbody>
</table>
20.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "CertifyCreation_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>key referenced by <code>signHandle</code> is not a signing key</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td><code>inScheme</code> is not compatible with <code>signHandle</code></td>
</tr>
<tr>
<td>TPM_RC_TICKET</td>
<td><code>creationTicket</code> does not match <code>objectHandle</code></td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>digest generated for <code>inScheme</code> is greater or has larger size than the modulus of <code>signHandle</code>, or the buffer for the result in <code>signature</code> is too small (for an RSA key); invalid commit status (for an ECC key with a split scheme).</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_CertifyCreation(
    CertifyCreation_In *in, // IN: input parameter list
    CertifyCreation_Out *out // OUT: output parameter list
)
{
    TPM_RC                  result;
    TPM2B_NAME              name;
    TPMT_TK_CREATION        ticket;
    TPMS_ATTEST             certifyInfo;

    // Input Validation
    // CertifyCreation specific input validation
    // Get certified object name
    name.t.size = ObjectGetName(in->objectHandle, &name.t.name);
    // Re-compute ticket
    TicketComputeCreation(in->creationTicket.hierarchy, &name,
                          &in->creationHash, &ticket);
    // Compare ticket
    if(!Memory2BEqual(&ticket.digest.b, &in->creationTicket.digest.b))
        return TPM_RC_TICKET + RC_CertifyCreation_creationTicket;

    // Command Output
    // Common fields
    result = FillInAttestInfo(in->signHandle, &in->inScheme, &in->qualifyingData, &certifyInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_CertifyCreation_signHandle;
        else
            return RcSafeAddToResult(result, RC_CertifyCreation_inScheme);
    }

    // CertifyCreation specific fields
    // Attestation type
    certifyInfo.type = TPM_ST_ATTEST_CREATION;
    certifyInfo.attested.creation.objectName = name;

    // Copy the creationHash
    certifyInfo.attested.creation.creationHash = in->creationHash;
```
47 // Sign attestation structure. A NULL signature will be returned if
48 // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
49 // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
50 // this point
51 result = SignAttestInfo(in->signHandle,
52 &in->inScheme,
53 &certifyInfo,
54 &in->qualifyingData,
55 &out->certifyInfo,
56 &out->signature);
57
58 // TPM_RC_ATTRIBUTES cannot be returned here as FillInAttestInfo would already
59 // have returned TPM_RC_KEY
60 pAssert(result != TPM_RC_ATTRIBUTES);
61
62 if(result != TPM_RC_SUCCESS)
63 return result;
64
65 // orderly state should be cleared because of the reporting of clock info
66 // if signing happens
67 if(in->signHandle != TPM_RH_NULL)
68 g_clearOrderly = TRUE;
69
70 return TPM_RC_SUCCESS;
71 }
20.4 TPM2_Quote

20.4.1 General Description

This command is used to quote PCR values.

NOTE See 20.1 for description of how the signing scheme is selected.

The TPM will hash the list of PCR selected by PCRselect using the hash algorithm associated with signHandle (this is the hash algorithm of the signing scheme, not the nameAlg of signHandle).

The digest is computed as the hash of the concatenation of all of the digest values of the selected PCR.

The concatenation of PCR is described in Part 1, Selecting Multiple PCR.
20.4.2 Command and Response

Table 81 — TPM2_Quote Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Quote</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@signHandle</td>
<td>handle of key that will perform signature</td>
</tr>
<tr>
<td>Auth Index: 1</td>
<td>Auth Role: USER</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>qualifyingData</td>
<td>data supplied by the caller</td>
</tr>
<tr>
<td>TPMT_SIG_SCHEME+</td>
<td>inScheme</td>
<td>signing scheme to use if the scheme for signHandle is TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPML_PCR_SELECTION</td>
<td>PCRselect</td>
<td>PCR set to quote</td>
</tr>
</tbody>
</table>

Table 82 — TPM2_Quote Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ATTEST</td>
<td>quoted</td>
<td>the quoted information</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>signature</td>
<td>the signature over quoted</td>
</tr>
</tbody>
</table>
20.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "Quote_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>signHandle does not reference a signing key;</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>the scheme is not compatible with sign key type, or input scheme is not compatible with default scheme, or the chosen scheme is not a valid sign scheme</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_Quote(
    Quote_In *in, // IN: input parameter list
    Quote_Out *out // OUT: output parameter list
)
{
    TPM_RC result;
    TPM_ALG_HASH hashAlg;
    TPMS_ATTEST quoted;

    // Command Output
    // Filling in attest information
    // Common fields
    // FillInAttestInfo will return TPM_RC_SCHEME or TPM_RC_KEY
    result = FillInAttestInfo(in->signHandle,
        &in->inScheme,
        &in->qualifyingData,
        &quoted);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_Quote_signHandle;
        else
            return RcSafeAddToResult(result, RC_Quote_inScheme);
    }

    // Quote specific fields
    // Attestation type
    quoted.type = TPM_ST_ATTEST_QUOTE;
    // Get hash algorithm in sign scheme. This hash algorithm is used to
    // compute PCR digest. If there is no algorithm, then the PCR cannot
    // be digested and this command returns TPM_RC_SCHEME
    hashAlg = in->inScheme.details.any.hashAlg;
    if(hashAlg == TPM_ALG_NULL)
        return TPM_RC_SCHEME + RC_Quote_inScheme;

    // Compute PCR digest
    PCRComputeCurrentDigest(hashAlg,
        &in->PCRselect,
        &quoted.attested.quote.pcrDigest);

    // Copy PCR select. "PCRselect" is modified in PCRComputeCurrentDigest
    // function
    quoted.attested.quote.pcrSelect = in->PCRselect;
```
// Sign attestation structure. A NULL signature will be returned if
// signHandle is TPM_RH_NULL. TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES
// error may be returned by SignAttestInfo.
// NOTE: TPM_RC_ATTRIBUTES means that the key is not a signing key but that
// was checked above and TPM_RC_KEY was returned. TPM_RC_VALUE means that the
// value to sign is too large but that means that the digest is too big and
// that can't happen.
result = SignAttestInfo(in->signHandle,
                        &in->inScheme,
                        &quoted,
                        &in->qualifyingData,
                        &out->quoted,
                        &out->signature);
if(result != TPM_RC_SUCCESS)
    return result;

// orderly state should be cleared because of the reporting of clock info
// if signing happens
if(in->signHandle != TPM_RH_NULL)
    g_clearOrderly = TRUE;
return TPM_RC_SUCCESS;
20.5 TPM2_GetSessionAuditDigest

20.5.1 General Description

This command returns a digital signature of the audit session digest.

NOTE 1 See 20.1 for description of how the signing scheme is selected.

If sessionHandle is not an audit session, the TPM shall return TPM_RC_TYPE.

NOTE 2 A session does not become an audit session until the successful completion of the command in which the session is first used as an audit session.

This command requires authorization from the privacy administrator of the TPM (expressed with endorsementAuth) as well as authorization to use the key associated with signHandle.

If this command is audited, then the audit digest that is signed will not include the digest of this command because the audit digest is only updated when the command completes successfully.

This command does not cause the audit session to be closed and does not reset the digest value.

NOTE 3 The audit session digest will be reset if the sessionHandle is used as the audit session for the command and the auditReset attribute of the session is set; and this command will be the first command in the audit digest.

NOTE 4 A reason for using 'sessionHahdle' in this command is so that the continueSession attribute may be CLEAR. This will flush the session at the end of the command.
### 20.5.2 Command and Response

#### Table 83 — TPM2_GetSessionAuditDigest Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_GetSessionAuditDigest</td>
</tr>
<tr>
<td>TPMI_RH_ENDORSEMENT</td>
<td>@privacyAdminHandle</td>
<td>handle of the privacy administrator (TPM_RH_ENDORSEMENT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>@signHandle</td>
<td>handle of the signing key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_SH_HMAC</td>
<td>sessionHandle</td>
<td>handle of the audit session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>qualifyingData</td>
<td>user-provided qualifying data – may be zero-length</td>
</tr>
<tr>
<td>TPMT_SIG_SCHHEME+</td>
<td>inScheme</td>
<td>signing scheme to use if the scheme for signHandle is TPM_ALG_NULL</td>
</tr>
</tbody>
</table>

#### Table 84 — TPM2_GetSessionAuditDigest Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ATTEST</td>
<td>auditInfo</td>
<td>the audit information that was signed</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>signature</td>
<td>the signature over auditInfo</td>
</tr>
</tbody>
</table>
20.5.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "GetSessionAuditDigest_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>key referenced by signHandle is not a signing key</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>inScheme is incompatible with signHandle type; or 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>
<tr>
<td>TPM_RC_TYPE</td>
<td>sessionHandle does not reference an audit session</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

TPM2_GetSessionAuditDigest(
  GetSessionAuditDigest_In *in,       // IN: input parameter list
  GetSessionAuditDigest_Out *out      // OUT: output parameter list
)
{
  TPM_RC result;
  SESSION *session;
  TPMS_ATTEST auditInfo;

  // Input Validation
  // SessionAuditDigest specific input validation
  // Get session pointer
  session = SessionGet(in->sessionHandle);
  // session must be an audit session
  if(session->attributes.isAudit == CLEAR)
    return TPM_RC_TYPE + RC_GetSessionAuditDigest_sessionHandle;

  // Command Output
  // Filling in attest information
  // Common fields
  result = FillInAttestInfo(in->signHandle,
                            &in->inScheme,
                            &in->qualifyingData,
                            &auditInfo);

  if(result != TPM_RC_SUCCESS)
    {
      if(result == TPM_RC_KEY)
        return TPM_RC_KEY + RC_GetSessionAuditDigest_signHandle;
      else
        return RcSafeAddToResult(result, RC_GetSessionAuditDigest_inScheme);
    }

  // SessionAuditDigest specific fields
  // Attestation type
  auditInfo.type = TPM_ST_ATTEST_SESSION_AUDIT;

  // Copy digest
```


```c
    auditInfo.attested.sessionAudit.sessionDigest = session->u2.auditDigest;

    // Exclusive audit session
    if (g_exclusiveAuditSession == in->sessionHandle)
        auditInfo.attested.sessionAudit.exclusiveSession = TRUE;
    else
        auditInfo.attested.sessionAudit.exclusiveSession = FALSE;

    // Sign attestation structure. A NULL signature will be returned if
    // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEMA or TPM_RC_ATTRIBUTES error may be returned at
    // this point
    result = SignAttestInfo(in->signHandle,
        &in->inScheme,
        &auditInfo,
        &in->qualifyingData,
        &out->auditInfo,
        &out->signature);
    if (result != TPM_RC_SUCCESS)
        return result;

    // orderly state should be cleared because of the reporting of clock info
    // if signing happens
    if (in->signHandle != TPM_RH_NULL)
        g_clearOrderly = TRUE;

    return TPM_RC_SUCCESS;
```
20.6 TPM2_GetCommandAuditDigest

20.6.1 General Description

This command returns the current value of the command audit digest, a digest of the commands being audited, and the audit hash algorithm. These values are placed in an attestation structure and signed with the key referenced by signHandle.

NOTE 1 See 20.1 for description of how the signing scheme is selected.

When this command completes successfully, and signHandle is not TPM_RH_NULL, the audit digest is cleared.

NOTE 2 The way that the TPM tracks that the digest is clear is vendor-dependent. The reference implementation resets the size of the digest to zero.

If this command is being audited, then the signed digest produced by the command will not include the command. At the end of this command, the audit digest will be extended with cpHash and the rpHash of the command which would change the command audit digest signed by the next invocation of this command.

This command requires authorization from the privacy administrator of the TPM (expressed with endorsementAuth) as well as authorization to use the key associated with signHandle.
### 20.6.2 Command and Response

**Table 85 — TPM2_GetCommandAuditDigest Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_GetCommandAuditDigest (NV)</td>
</tr>
<tr>
<td>TPMI_RH_ENDORSEMENT</td>
<td>@privacyHandle</td>
<td>handle of the privacy administrator (TPM_RH_ENDORSEMENT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPML_DH_OBJECT+</td>
<td>@signHandle</td>
<td>the handle of the signing key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>qualifyingData</td>
<td>other data to associate with this audit digest</td>
</tr>
<tr>
<td>TPMT_SIG_SCHEME+</td>
<td>inScheme</td>
<td>signing scheme to use if the scheme for signHandle is TPM_ALG_NULL</td>
</tr>
</tbody>
</table>

**Table 86 — TPM2_GetCommandAuditDigest Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_MST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ATTEST</td>
<td>auditInfo</td>
<td>the auditInfo that was signed</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>signature</td>
<td>the signature over auditInfo</td>
</tr>
</tbody>
</table>
20.6.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "GetCommandAuditDigest_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_KEY</td>
<td>key referenced by signHandle is not a signing key</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>inScheme is incompatible with signHandle type; or 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>
<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

TPM2_GetCommandAuditDigest(
    GetCommandAuditDigest_In *in,  // IN: input parameter list
    GetCommandAuditDigest_Out *out   // OUT: output parameter list
)
{
    TPM_RC result;
    TPMS_ATTEST auditInfo;

    // Command Output

    // Filling in attest information
    // Common fields
    result = FillInAttestInfo(in->signHandle,
        in->inScheme,
        in->qualifyingData,
        &auditInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_GetCommandAuditDigest_signHandle;
        else
            return RcSafeAddToResult(result, RC_GetCommandAuditDigest_inScheme);
    }

    // CommandAuditDigest specific fields
    // Attestation type
    auditInfo.type = TPM_ST_ATTEST_COMMAND_AUDIT;

    // Copy audit hash algorithm
    auditInfo.attested.commandAudit.digestAlg = gp.auditHashAlg;

    // Copy counter value
    auditInfo.attested.commandAudit.auditCounter = gp.auditCounter;

    // Copy command audit log
    auditInfo.attested.commandAudit.auditDigest = gr.commandAuditDigest;
    CommandAuditGetDigest(&auditInfo.attested.commandAudit.commandDigest);

    // Sign attestation structure. A NULL signature will be returned if
    // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
    // this point
```
```c
result = SignAttestInfo(in->signHandle,
                        &in->inScheme,
                        &auditInfo,
                        &in->qualifyingData,
                        &out->auditInfo,
                        &out->signature);

if(result != TPM_RC_SUCCESS)
    return result;

// Internal Data Update

if(in->signHandle != TPM_RH_NULL)
{
    // Reset log
    gr.commandAuditDigest.t.size = 0;
    
    // orderly state should be cleared because of the update in
    // commandAuditDigest, as well as the reporting of clock info
    g_clearOrderly = TRUE;
}

return TPM_RC_SUCCESS;
```
20.7 TPM2_GetTime

20.7.1 General Description

This command returns the current values of Time and Clock.

NOTE 1 See 20.1 for description of how the signing scheme is selected.

The values of Clock, resetCount and restartCount appear in two places in timeInfo: once in TPMS_ATTEST.clockInfo and again in TPMS_ATTEST.attested.time.clockInfo. The firmware version number also appears in two places (TPMS_ATTEST.firmwareVersion and TPMS_ATTEST.attested.time.firmwareVersion). If signHandle is in the endorsement or platform hierarchies, both copies of the data will be the same. However, if signHandle is in the storage hierarchy or is TPM_RH_NULL, the values in TPMS_ATTEST.clockInfo and TPMS_ATTEST.firmwareVersion are obfuscated but the values in TPM_ATTEST.attested.time are not.

NOTE 2 The purpose of this duplication is to allow an entity who is trusted by the privacy Administrator to correlate the obfuscated values with the clear-text values.
### 20.7.2 Command and Response

**Table 87 — TPM2_GetTime Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_GetTime</td>
</tr>
<tr>
<td>TPMI_RH_ENDORSEMENT</td>
<td>@privacyAdminHandle</td>
<td>handle of the privacy administrator (TPM_RH_ENDORSEMENT)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>@signHandle</td>
<td>the keyHandle identifier of a loaded key that can perform digital signatures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>qualifyingData</td>
<td>data to tick stamp</td>
</tr>
<tr>
<td>TPMT_SIG_SCHEME+</td>
<td>inScheme</td>
<td>signing scheme to use if the scheme for signHandle is TPM_ALG_NULL</td>
</tr>
</tbody>
</table>

**Table 88 — TPM2_GetTime Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td>.</td>
</tr>
<tr>
<td>TPM2B_ATTEST</td>
<td>timeInfo</td>
<td>standard TPM-generated attestation block</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>signature</td>
<td>the signature over timeInfo</td>
</tr>
</tbody>
</table>
20.7.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "GetTime_fp.h"

Error Returns | Meaning
---|---
TPM_RC_KEY | key referenced by signHandle is not a signing key
TPM_RC_SCHEME | inScheme is incompatible with signHandle type; or 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
TPM_RC_VALUE | 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)

TPM_RC
TPM2_GetTime(
    GetTime_In  *in,    // IN: input parameter list
    GetTime_Out *out    // OUT: output parameter list
)
{
    TPM_RC                  result;
    TPMS_ATTEST             timeInfo;

    // Command Output
    // Filling in attest information
    // Common fields
    result = FillInAttestInfo(in->signHandle,
                              &in->inScheme,
                              &in->qualifyingData,
                              &timeInfo);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_GetTime_signHandle;
        else
            return RcSafeAddToResult(result, RC_GetTime_inScheme);
    }

    // GetClock specific fields
    // Attestation type
    timeInfo.type = TPM_ST_ATTEST_TIME;

    // current clock in plain text
    timeInfo.attested.time.time.time = g_time;
    TimeFillInfo(&timeInfo.attested.time.time.clockInfo);

    // Firmware version in plain text
    timeInfo.attested.time.firmwareVersion
        = ((UINT64) gp.firmwareV1) << 32;
    timeInfo.attested.time.firmwareVersion += gp.firmwareV2;

    // Sign attestation structure. A NULL signature will be returned if
    // signHandle is TPM_RH_NULL. A TPM_RC_NV_UNAVAILABLE, TPM_RC_NV_RATE,
    // TPM_RC_VALUE, TPM_RC_SCHEME or TPM_RC_ATTRIBUTES error may be returned at
    // this point
    result = SignAttestInfo(in->signHandle,
```

Family “2.0”

Published

Level 00 Revision 00.99

Copyright © TCG 2006-2013

October 31, 2013
if (result != TPM_RC_SUCCESS)
    return result;

// orderly state should be cleared because of the reporting of clock info
// if signing happens
if (in->signHandle != TPM_RH_NULL)
    g_clearOrderly = TRUE;

return TPM_RC_SUCCESS;
21 Ephemeral EC Keys

21.1 Introduction

The TPM generates keys that have different lifetimes. TPM keys in a hierarchy can be persistent for as long as the seed of the hierarchy is unchanged and these keys may be used multiple times. Other TPM-generated keys are only useful for a single operation. Some of these single-use keys are used in the command in which they are created. Examples of this use are TPM2_Duplicate() where an ephemeral key is created for a single pass key exchange with another TPM. However, there are other cases, such as anonymous attestation, where the protocol requires two passes where the public part of the ephemeral key is used outside of the TPM before the final command "consumes" the ephemeral key.

For these uses, TPM2_Commit() or TPM2_EC_Ephemeral() may be used to have the TPM create an ephemeral EC key and return the public part of the key for external use. Then in a subsequent command, the caller provides a reference to the ephemeral key so that the TPM can retrieve or recreate the associated private key.

When an ephemeral EC key is created, it is assigned a number and that number is returned to the caller as the identifier for the key. This number is not a handle. A handle is assigned to a key that may be context saved but these ephemeral EC keys may not be saved and do not have a full key context. When a subsequent command uses the ephemeral key, the caller provides the number of the ephemeral key. The TPM uses that number to either look up or recompute the associated private key. After the key is used, the TPM records the fact that the key has been used so that it cannot be used again.

As mentioned, the TPM can keep each assigned private ephemeral key in memory until it is used. However, this could consume a large amount of memory. To limit the memory size, the TPM is allowed to restrict the number of pending private keys – keys that have been allocated but not used.

NOTE The minimum number of ephemeral keys is determined by a platform specific specification

To further reduce the memory requirements for the ephemeral private keys, the TPM is allowed to use pseudo-random values for the ephemeral keys. Instead of keeping the full value of the key in memory, the TPM can use a counter as input to a KDF. Incrementing the counter will cause the TPM to generate a new pseudo-random value.

Using the counter to generate pseudo-random private ephemeral keys greatly simplifies tracking of key usage. When a counter value is used to create a key, a bit in an array may be set to indicate that the key use is pending. When the ephemeral key is consumed, the bit is cleared. This prevents the key from being used more than once.

Since the TPM is allowed to restrict the number of pending ephemeral keys, the array size can be limited. For example, a 128 bit array would allow 128 keys to be "pending".

The management of the array is described in greater detail in the Split Operations clause in Annex C of part 1.
21.2 TPM2_Commit

21.2.1 General Description

TPM2_Commit() performs the first part of an ECC anonymous signing operation. The TPM will perform the point multiplications on the provided points and return intermediate signing values. The signHandle parameter shall refer to an ECC key with the sign attribute (TPM_RC_ATTRIBUTES) using an anonymous signing scheme (TPM_RC_SCHEME).

For this command, \( p1, s2 \) and \( y2 \) are optional parameters. If \( s2 \) is an Empty Buffer, then the TPM shall return TPM_RC_SIZE if \( y2 \) is not an Empty Buffer. If \( p1, s2, \) and \( y2 \) are all Empty Buffers, the TPM shall return TPM_RC_NO_RESULT.

In the algorithm below, the following additional values are used in addition to the command parameters:

- \( H_{\text{nameAlg}} \): hash function using the \text{nameAlg} of the key associated with signHandle
- \( p \): field modulus of the curve associated with signHandle
- \( n \): order of the curve associated with signHandle
- \( d_s \): private key associated with signHandle
- \( c \): counter that increments each time a TPM2_Commit() is successfully completed
- \( A[i] \): array of bits used to indicate when a value of \( c \) has been used in a signing operation; values of \( i \) are 0 to \( 2n-1 \)
- \( k \): nonce that is set to a random value on each TPM Reset; nonce size is twice the security strength of any ECDAA key supported by the TPM.

The algorithm is:

a) set \( K, L, \) and \( E \) to be Empty Buffers.

b) if \( s2 \) is not an Empty Buffer, compute \( x2 \leftarrow H_{\text{nameAlg}}(s2) \mod p \), else skip to step (e)

c) if \( (x2, y2) \) is not a point on the curve of signHandle, return TPM_RC_ECC_POINT

d) set \( K \leftarrow [d_s](x2, y2) \)

e) generate or derive \( r \) (see the "Commit Random Value" clause in Part 1)

f) set \( r \leftarrow r \mod n \)

\textbf{NOTE 1} \quad nLen \ is \ the \ number \ of \ bits \ in \ n

g) if \( p1 \) is an Empty Buffer, skip to step i)

h) if \( (p1) \) is not a point on the curve of signHandle, return TPM_RC_ECC_POINT

i) set \( E \leftarrow [r](p1) \)

j) if \( K \) is not an Empty Buffer, set \( L \leftarrow [r](x2, y2) \)

k) if \( K, L, \) or \( E \) is the point at infinity, return TPM_RC_NO_RESULT

l) set \( \text{counter} \leftarrow \text{commitCount} \)

m) set \( \text{commitCount} \leftarrow \text{commitCount} + 1 \)
NOTE 2  Depending on the method of generating r, it may be necessary to update the tracking array here.

n) output $K, L, E$ and $counter$

NOTE 3  Depending on the input parameters K and L may be Empty Buffers or E may be an Empty Buffer.
21.2.2 Command and Response

**Table 89 — TPM2Commit Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>paramSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_Commit handle of the key that will be used in the signing operation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>@signHandle</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>P1</td>
<td>a point (M) on the curve used by (\text{signHandle})</td>
</tr>
<tr>
<td>TPM2B_SENSITIVE_DATA</td>
<td>s2</td>
<td>octet array used to derive x-coordinate of a base point</td>
</tr>
<tr>
<td>TPM2B_ECC_PARAMETER</td>
<td>y2</td>
<td>y coordinate of the point associated with (s2)</td>
</tr>
</tbody>
</table>

**Table 90 — TPM2Commit Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>paramSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>K</td>
<td>ECC point (K := [d_s](x_2, y_2))</td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>L</td>
<td>ECC point (L := [r](x_2, y_2))</td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>E</td>
<td>ECC point (E := [r]P_1)</td>
</tr>
<tr>
<td>UINT16</td>
<td>counter</td>
<td>least-significant 16 bits of (\text{commitCount})</td>
</tr>
</tbody>
</table>
21.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Commit_fp.h"
#ifdef TPM_ALG_ECC

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>keyHandle references a restricted key that is not a signing key</td>
</tr>
<tr>
<td>TPM_RC_ECC_POINT</td>
<td>either P1 or the point derived from s2 is not on the curve of keyHandle</td>
</tr>
<tr>
<td>TPM_RC_HASH</td>
<td>invalid name algorithm in keyHandle</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>keyHandle does not reference an ECC key</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>keyHandle references a restricted signing key that does not use and</td>
</tr>
<tr>
<td></td>
<td>anonymous scheme</td>
</tr>
<tr>
<td>TPM_RC_NO_RESULT</td>
<td>K, L or E was a point at infinity; or failed to generate r value</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>s2 is empty but y2 is not or s2 provided but y2 is not</td>
</tr>
</tbody>
</table>

TPM_RC_Commit(
    Commit_In     *in,  // IN: input parameter list
    Commit_Out    *out   // OUT: output parameter list
)
{
    Object                  *eccKey;
    TPMS_ECC_POINT          P2;
    TPMS_ECC_POINT          *pP2 = NULL;
    TPMS_ECC_POINT          *pP1 = NULL;
    TPM2B_ECC_PARAMETER     r;
    TPM2B                   *p;
    TPM_RC                  result;
    UINT16                  hashResults;

    // Input Validation
    eccKey = ObjectGet(in->signHandle);

    // Input key must be an ECC key
    if(eccKey->publicArea.type != TPM_ALG_ECC)
        return TPM_RC_KEY + RC_Commit_signHandle;

    // if the key is restricted, it must be a signing key using an anonymous scheme
    if(eccKey->publicArea.objectAttributes.restricted == SET)
    {
        if(eccKey->publicArea.objectAttributes.sign != SET)
            return TPM_RC_ATTRIBUTES + RC_Commit_signHandle;
        if(!CryptIsSchemeAnonymous(
            eccKey->publicArea.parameters.eccDetail.scheme.scheme))
            return TPM_RC_SCHEME + RC_Commit_signHandle;
    }
    else
    {
        // if not restricted, s2, and y2 must be an Empty Buffer
        if((in->s2.t.size) != (in->y2.t.size == 0))
            return TPM_RC_SIZE + RC_Commit_s2;
    }

    // Make sure that both parts of P2 are present if either is present
    if((in->s2.t.size == 0) != (in->y2.t.size == 0))
```
return TPM_RC_SIZE + RC_Commit_y2;

// Get prime modulus for the curve. This is needed later but getting this now
// allows confirmation that the curve exists
p = (TPM2B *)CryptEccGetParameter('p',
    eccKey->publicArea.parameters.eccDetail.curveID);

// if no p, then the curve ID is bad
// NOTE: This should never occur if the input unmarshaling code is working
correctly
if (p == NULL)
    return TPM_RC_KEY + RC_Commit_signHandle;

// Get the random value that will be used in the point multiplications
// Note: this does not commit the count.
if (!CryptGenerateR(&r, NULL,
    eccKey->publicArea.parameters.eccDetail.curveID, &eccKey->name))
    return TPM_RC_NO_RESULT;

// Set up P2 if s2 and Y2 are provided
if (in->s2.t.size != 0)
{
    pP2 = &P2;

    // copy y2 for P2
    MemoryCopy2B(&P2.y.b, &in->y2.b, sizeof(P2.y.t.buffer));
     // Compute x2 HnameAlg(s2) mod p
    //      do the hash operation on s2 with the size of curve 'p'
    hashResults = CryptHashBlock(eccKey->publicArea.nameAlg,
        in->s2.t.size,
        in->s2.t.buffer,
        p->size,
        P2.x.t.buffer);

    // If there were error returns in the hash routine, indicate a problem
    // with the hash in
    if (hashResults == 0)
        return TPM_RC_HASH + RC_Commit_signHandle;

    // set the size of the X value to the size of the hash
    P2.x.t.size = hashResults;

    // set p2.x = hash(s2) mod p
    if (CryptDivide(&P2.x.b, p, NULL, &P2.x.b) != TPM_RC_SUCCESS)
        return TPM_RC_NO_RESULT;

    if (!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID,
        pP2))
        return TPM_RC_ECC_POINT + RC_Commit_s2;
    
    if (eccKey->attributes.publicOnly == SET)
        return TPM_RC_KEY + RC_Commit_signHandle;
}
else
{
    // If there is a P1, make sure that it is on the curve
    // NOTE: an "empty" point has two UINT16 values which are the size values
    // for each of the coordinates.
    if (in->P1.t.size > 4)
108  pP1 = &in->P1.t.point;
109  if (!CryptEccIsPointOnCurve(eccKey->publicArea.parameters.eccDetail.curveID, 
110     pP1))
111      return TPM_RC_ECC_POINT + RC_Commit_P1;
112  }
113
114  // Pass the parameters to CryptCommit.
115  // The work is not done inline because it does several point multiplies
116  // with the same curve. There is significant optimization by not
117  // having to reload the curve parameters multiple times.
118  result = CryptCommitCompute(&out->K.t.point,
119      &out->L.t.point,
120      &out->E.t.point,
121      eccKey->publicArea.parameters.eccDetail.curveID,
122      pP1,
123      pP2,
124      &eccKey->sensitive.sensitive.ecc,
125      &r);
126  if (result != TPM_RC_SUCCESS)
127     return result;
128
129  out->K.t.size = TPMS_ECC_POINT_Marshal(&out->K.t.point, NULL, NULL);
130  out->L.t.size = TPMS_ECC_POINT_Marshal(&out->L.t.point, NULL, NULL);
131  out->E.t.size = TPMS_ECC_POINT_Marshal(&out->E.t.point, NULL, NULL);
132
133  // The commit computation was successful so complete the commit by setting
134  // the bit
135  out->counter = CryptCommit();
136
137  return TPM_RC_SUCCESS;
138 } #endif
21.3 TPM2_EC_Ephemeral

21.3.1 General Description

TPM2_EC_Ephemeral() creates an ephemeral key for use in a two-phase key exchange protocol. The TPM will use the commit mechanism to assign an ephemeral key $r$ and compute a public point $Q := [r]G$ where $G$ is the generator point associated with $curvelD$. 
21.3.2 Command and Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>paramSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_EC_Ephemeral</td>
</tr>
<tr>
<td>TPM_ECC_CURVE</td>
<td>curveID</td>
<td>The curve for the computed ephemeral point</td>
</tr>
</tbody>
</table>

Table 91 — TPM2_EC_Ephemeral Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>paramSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ECC_POINT</td>
<td>Q</td>
<td>ephemeral public key $Q := [r]G$</td>
</tr>
<tr>
<td>UINT16</td>
<td>counter</td>
<td>least-significant 16 bits of ( \text{commitCount} )</td>
</tr>
</tbody>
</table>

Table 92 — TPM2_EC_Ephemeral Response
21.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "EC_Ephemeral_fp.h"
#ifdef TPM_ALG_ECC

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>...</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_EC_Ephemeral(
    EC_Ephemeral_In *in, // IN: input parameter list
    EC_Ephemeral_Out *out // OUT: output parameter list
)
{
    TPM2B_ECC_PARAMETER r;

    // Get the random value that will be used in the point multiplications
    // Note: this does not commit the count.
    if(!CryptGenerateR(&r,
        NULL,
        in->curveID,
        NULL))
        return TPM_RC_NO_RESULT;
    CryptEccPointMultiply(&out->Q.t.point, in->curveID, &r, NULL);
    // commit the count value
    out->counter = CryptCommit();
    return TPM_RC_SUCCESS;
}
#endif
```
22 Signing and Signature Verification

22.1 TPM2_VerifySignature

22.1.1 General Description

This command uses loaded keys to validate a signature on a message with the message digest passed to the TPM.

If the signature check succeeds, then the TPM will produce a TPMT_TK_VERIFIED. Otherwise, the TPM shall return TPM_RC_SIGNATURE.

NOTE 1 A valid ticket may be used in subsequent commands to provide proof to the TPM that the TPM has validated the signature over the message using the key referenced by keyHandle.

If keyHandle references an asymmetric key, only the public portion of the key needs to be loaded. If keyHandle references a symmetric key, both the public and private portions need to be loaded.

NOTE 2 The sensitive area of the symmetric object is required to allow verification of the symmetric signature (the HMAC).
### 22.1.2 Command and Response

#### Table 93 — TPM2_VerifySignature Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_VerifySignature</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>keyHandle</td>
<td>handle of public key that will be used in the validation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>digest</td>
<td>digest of the signed message</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>signature</td>
<td>signature to be tested</td>
</tr>
</tbody>
</table>

#### Table 94 — TPM2_VerifySignature Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMT_TK_VERIFIED</td>
<td>validation</td>
<td></td>
</tr>
</tbody>
</table>
### Detailed Actions

```c
#include "InternalRoutines.h"
#include "VerifySignature_fp.h"

<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 signing key</td>
</tr>
<tr>
<td>TPM_RC_SIGNATURE</td>
<td>signature is not genuine</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>CryptVerifySignature()</td>
</tr>
<tr>
<td>TPM_RC_HANDLE</td>
<td>the input handle is not a sign key with private portion loaded</td>
</tr>
</tbody>
</table>

```TPM_RC

```c
TPM2_VerifySignature(
    VerifySignature_In *in,       // IN: input parameter list
    VerifySignature_Out *out      // OUT: output parameter list
);```

```c
 TPM_RC                   result;
TPM2B_NAME               name;
OBJECT                   *signObject;
TPMI_RH_HIERARCHY        hierarchy;

// Input Validation

// Get sign object pointer
signObject = ObjectGet(in->keyHandle);

// The object to validate the signature must be a signing key.
if(signObject->publicArea.objectAttributes.sign != SET)
    return TPM_RC_ATTRIBUTES + RC_VerifySignature_keyHandle;

// If it doesn't have a sensitive area loaded
// then it can't be a keyed hash signing key
if(signObject->attributes.publicOnly == SET
    && signObject->publicArea.type == TPM_ALG_KEYEDHASH)
    return TPM_RC_HANDLE + RC_VerifySignature_keyHandle;

// Validate Signature.  A TPM_RC_BINDING, TPM_RC_SCHEME or TPM_RC_SIGNATURE
// error may be returned by CryptCVerifySignature()
result = CryptVerifySignature(in->keyHandle, &in->digest, &in->signature);
if(result != TPM_RC_SUCCESS)
    return RcSafeAddToResult(result, RC_VerifySignature_signature);

// Command Output

hierarchy = ObjectGetHierarchy(in->keyHandle);
if (hierarchy == TPM_RH_NULL
    || signObject->publicArea.nameAlg == TPM_ALG_NULL)
    {
        // produce empty ticket if hierarchy is TPM_RH_NULL or nameAlg is
        // TPM_ALG_NULL
        out->validation.tag = TPM_ST_VERIFIED;
        out->validation.hierarchy = TPM_RH_NULL;
        out->validation.digest.t.size = 0;
    } else
    {
```
// Get object name that verifies the signature
name.t.size = ObjectGetName(in->keyHandle, &name.t.name);
// Compute ticket
TicketComputeVerified(hierarchy, &in->digest, &name, &out->validation);
}

return TPM_RC_SUCCESS;
22.2 TPM2_Sign

22.2.1 General Description

This command causes the TPM to sign an externally provided hash with the specified asymmetric signing key.

NOTE 1 Symmetric “signing” is done with an HMAC.

If keyHandle references a restricted signing key, then validation shall be provided indicating that the TPM performed the hash of the data and validation shall indicate that hashed data did not start with TPM_GENERATED_VALUE.

NOTE 2 If the hashed data did start with TPM_GENERATED_VALUE, then the validation will be a NULL ticket.

If the scheme of keyHandle is not TPM_ALG_NULL, then inScheme shall either be the same scheme as keyHandle or TPM_ALG_NULL.

If the scheme of keyHandle is TPM_ALG_NULL, the TPM will sign using inScheme; otherwise, it will sign using the scheme of keyHandle.

NOTE 3 When the signing scheme requires a hash algorithm, the hash is defined in the qualifying data of the scheme.

If inScheme is not a valid signing scheme for the type of keyHandle (or TPM_ALG_NULL), then the TPM shall return TPM_RC_SCHEME.

If the scheme of keyHandle is an anonymous scheme, then inScheme shall have the same scheme algorithm as keyHandle and inScheme will contain a counter value that will be used in the signing process.

As long as it is no larger than allowed, the digest parameter is not required to have any specific size but the signature operation may fail if digest is too large for the selected scheme.

If the validation parameter is not the Empty Buffer, then it will be checked even if the key referenced by keyHandle is not a restricted signing key.
22.2.2 Command and Response

<table>
<thead>
<tr>
<th>Table 95 — TPM2_Sign Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_CC</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
</tr>
<tr>
<td>Auth Index: 1</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
</tr>
<tr>
<td>TPMT_SIG_SCHEME</td>
</tr>
<tr>
<td>TPMT_TK_HASHCHECK</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 96 — TPM2_Sign Response</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TPM_ST</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_RC</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
</tr>
</tbody>
</table>
22.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Sign_fp.h"
#include "Attest_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>key referenced by <code>keHandle</code> is not a signing key</td>
</tr>
<tr>
<td>TPM_RC_BINDING</td>
<td>The public and private portions of the key are not properly bound.</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td><code>inScheme</code> is not compatible with <code>keyHandle</code>; both <code>inScheme</code> and key's</td>
</tr>
<tr>
<td></td>
<td>default scheme are empty; or <code>inScheme</code> is empty while key's default</td>
</tr>
<tr>
<td></td>
<td>scheme requires explicit input scheme (split signing); or non-empty</td>
</tr>
<tr>
<td></td>
<td>default key scheme differs from <code>inScheme</code></td>
</tr>
<tr>
<td>TPM_RC_TICKET</td>
<td><code>validation</code> is not a valid ticket</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>the value to sign is larger than allowed for the type of <code>keyHandle</code></td>
</tr>
</tbody>
</table>

```

```

```c
TPM_RC

TPM2_Sign(
    Sign_In *in, // IN: input parameter list
    Sign_Out *out // OUT: output parameter list
)
{
    TPM_RC result;
    TPM_TK_HASHCHECK ticket;
    OBJECT *signKey;

    // Input Validation
    // Get sign key pointer
    signKey = ObjectGet(in->keyHandle);

    // If validation is provided, or the key is restricted, check the ticket
    if (in->validation.digest.t.size != 0 || signKey->publicArea.objectAttributes.restricted == SET)
    {
        // Compute and compare ticket
        TicketComputeHashCheck(in->validation.hierarchy, &in->digest, &ticket);

        if (!Memory2BEqual(&in->validation.digest.b, &ticket.digest.b))
            return TPM_RC_TICKET + RC_Sign_validation;
    }

    // Command Output

    // pick a scheme for sign. If the input sign scheme is not compatible with
    // the default scheme, return an error.
    result = CryptSelectSignScheme(in->keyHandle, &in->inScheme);
    if (result != TPM_RC_SUCCESS)
    {
        if (result == TPM_RC_KEY)
            return TPM_RC_KEY + RC_Sign_keyHandle;
        else
            return RcSafeAddToResult(result, RC_Sign_inScheme);
    }

    // Sign the hash. A TPM_RC_VALUE, TPM_RC_SCHEME, or TPM_RC_ATTRIBUTES
    // error may be returned at this point
    result = CryptSign(in->keyHandle, &in->inScheme, &in->digest, &out->signature);
```
45            return result;
46        }
47        

23 Command Audit

23.1 Introduction

If a command has been selected for command audit, the command audit status will be updated when that command completes successfully. The digest is updated as:

\[
\text{commandAuditDigest}_{\text{new}} := H_{\text{auditAlg}}(\text{commandAuditDigest}_{\text{old}} || \text{cpHash} || \text{rpHash})
\]

where

- \(H_{\text{auditAlg}}\) is the hash function using the algorithm of the audit sequence
- \(\text{commandAuditDigest}\) is the accumulated digest
- \(\text{cpHash}\) is the command parameter hash
- \(\text{rpHash}\) is the response parameter hash

TPM2_Shutdown() cannot be audited but TPM2_Startup() can be audited. If the \(\text{cpHash}\) of the TPM2_Startup() is TPM_SU_STATE, that would indicate that a TPM2_Shutdown() had been successfully executed.

TPM2_SetCommandCodeAuditStatus() is always audited.

If the TPM is in Failure mode, command audit is not functional.
23.2 TPM2_SetCommandCodeAuditStatus

23.2.1 General Description

This command may be used by the Privacy Administrator or platform to change the audit status of a command or to set the hash algorithm used for the audit digest, but not both at the same time.

If the auditAlg parameter is a supported hash algorithm and not the same as the current algorithm, then the TPM will check both setList and clearList are empty (zero length). If so, then the algorithm is changed, and the audit digest is cleared. If auditAlg is TPM_ALG_NULL or the same as the current algorithm, then the algorithm and audit digest are unchanged and the setList and clearList will be processed.

NOTE 1 Because the audit digest is cleared, the audit counter will increment the next time that an audited command is executed.

Use of TPM2_SetCommandCodeAuditStatus() to change the list of audited commands is an audited event. If TPM_CC_SetCommandCodeAuditStatus is in clearList, it is ignored.

NOTE 2 Use of this command to change the audit hash algorithm is not audited and the digest is reset when the command completes. The change in the audit hash algorithm is the evidence that this command was used to change the algorithm.

The commands in setList indicate the commands that to be added to the list of audited commands and the commands in clearList indicate the commands that will no longer be audited. It is not an error if a command in setList is already audited or is not implemented. It is not an error if a command in clearList is not currently being audited or is not implemented.

If a command code is in both setList and clearList, then it will not be audited (that is, setList shall be processed first).
### 23.2.2 Command and Response

**Table 97 — TPM2_SetCommandCodeAuditStatus Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_SetCommandCodeAuditStatus {NV}</td>
</tr>
<tr>
<td>TPMI_RH_PROVISION</td>
<td>@auth</td>
<td>Auth Index: 1, Auth Role: USER</td>
</tr>
<tr>
<td>TPM_ALG_HASH+</td>
<td>auditAlg</td>
<td>hash algorithm for the audit digest; if TPM_ALG_NULL, then the hash is not changed</td>
</tr>
<tr>
<td>TPML_CC</td>
<td>setList</td>
<td>list of commands that will be added to those that will be audited</td>
</tr>
<tr>
<td>TPML_CC</td>
<td>clearList</td>
<td>list of commands that will no longer be audited</td>
</tr>
</tbody>
</table>

**Table 98 — TPM2_SetCommandCodeAuditStatus Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
23.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "SetCommandCodeAuditStatus_fp.h"

TPM_RC
TPM2_SetCommandCodeAuditStatus(
SetCommandCodeAuditStatus_In   *in   // IN: input parameter list
)
{

TPM_RC result;
UINT32 i;
BOOL  changed = FALSE;

// The command needs NV update. Check if NV is available.
// A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
// this point
result = NvIsAvailable();
if (result != TPM_RC_SUCCESS)
    return result;

// Internal Data Update

// Update hash algorithm
if (in->auditAlg != TPM_ALG_NULL
    && in->auditAlg != gp.auditHashAlg)
{
    // Can't change the algorithm and command list at the same time
    if((in->setList.count != 0 || in->clearList.count != 0)
        return TPM_RC_VALUE + RC_SetCommandCodeAuditStatus_auditAlg;

    // Change the hash algorithm for audit
    gp.auditHashAlg = in->auditAlg;

    // Set the digest size to a unique value that indicates that the digest
    // algorithm has been changed. The size will be cleared to zero in the
    // command audit processing on exit.
    gr.commandAuditDigest.t.size = 1;

    // Save the change of command audit data (this sets g_updateNV so that NV
    // will be updated on exit.)
    NvWriteReserved(NV_AUDIT_HASH_ALG, &gp.auditComands);
}
else
{

    // Process set list
    for(i = 0; i < in->setList.count; i++)
        // If change is made in CommandAuditSet, set changed flag
        if(CommandAuditSet(in->setList.commandCodes[i]))
            changed = TRUE;

    // Process clear list
    for(i = 0; i < in->clearList.count; i++)
        // If change is made in CommandAuditClear, set changed flag
        if(CommandAuditClear(in->clearList.commandCodes[i]))
            changed = TRUE;

    // if change was made to command list, update NV
    if(changed)
        // this sets g_updateNV so that NV will be updated on exit.
        NvWriteReserved(NV_AUDIT_COMMANDS, &gp.auditComands);
```
return TPM_RC_SUCCESS;
}
}
24 Integrity Collection (PCR)

24.1 Introduction

In TPM 1.2, an Event was hashed using SHA-1 and then the 20-octet digest was extended to a PCR using TPM_Extend(). This specification allows the use of multiple PCR at a given Index, each using a different hash algorithm. Rather than require that the external software generate multiple hashes of the Event with each being extended to a different PCR, the Event data may be sent to the TPM for hashing. This ensures that the resulting digests will properly reflect the algorithms chosen for the PCR even if the calling software is unable to implement the hash algorithm.

NOTE 1 There is continued support for software hashing of events with TPM2_PCR_Extend().

To support recording of an Event that is larger than the TPM input buffer, the caller may use the command sequence described in clause 1.

Change to a PCR requires authorization. The authorization may be with either an authorization value or an authorization policy. The platform-specific specifications determine which PCR may be controlled by policy. All other PCR are controlled by authorization.

If a PCR may be associated with a policy, then the algorithm ID of that policy determines whether the policy is to be applied. If the algorithm ID is not TPM_ALG_NULL, then the policy digest associated with the PCR must match the $policySession\rightarrow policyDigest$ in a policy session. If the algorithm ID is TPM_ALG_NULL, then no policy is present and the authorization requires an EmptyAuth.

If a platform-specific specification indicates that PCR are grouped, then all the PCR in the group use the same authorization policy or authorization value.

$PcrUpdateCounter$ counter will be incremented on the successful completion of any command that modifies (Extends or resets) a PCR unless the platform-specific specification explicitly excludes the PCR from being counted.

NOTE 2 If a command causes PCR in multiple banks to change, the PCR Update Counter may be incremented either once or once for each bank.

A platform-specific specification may designate a set of PCR that are under control of the TCB. These PCR may not be modified without the proper authorization. Updates of these PCR shall not cause the PCR Update Counter to increment.

EXAMPLE Updates of the TCB PCR will not cause the PCR update counter to increment because these PCR are changed at the whim of the TCB and are not intended to represent the trust state of the platform.
24.2 TPM2_PCR_Extend

24.2.1 General Description

This command is used to cause an update to the indicated PCR. The digests parameter contains one or more tagged digest value identified by an algorithm ID. For each digest, the PCR associated with pcrHandle is Extended into the bank identified by the tag (hashAlg).

EXAMPLE

A SHA1 digest would be Extended into the SHA1 bank and a SHA256 digest would be Extended into a SHA256 bank.

For each list entry, the TPM will check to see if pcrNum is implemented for that algorithm. If so, the TPM shall perform the following operation:

\[
PCR.digest_{\text{new}}[pcrNum][\text{alg}] := H_{\text{alg}}(PCR.digest_{\text{old}}[pcrNum][\text{alg}] \parallel data[\text{alg}].\text{buffer})
\]

(6)

where

- \(H_{\text{alg}}()\) is the hash function using the hash algorithm associated with the PCR instance
- \(PCR.digest\) is the digest value in a PCR
- \(pcrNum\) is the PCR numeric selector (equal to pcrHandle - TPM_RH_PCR0)
- \(\text{alg}\) is the PCR algorithm selector for the digest
- \(data[\text{alg}].\text{buffer}\) is the bank-specific data to be extended

If no digest value is specified for a bank, then the PCR in that bank are not modified.

NOTE 1 This allows consistent operation of the digests list for all of the Event recording commands.

If a digest is present and the PCR in that bank is not implemented, the digest value is not used.

NOTE 2 If the caller includes digests for algorithms that are not implemented, then the TPM will fail the call because the unmarshalling of digests will fail. Each of the entries in the list is a TPMT_HA which is a hash algorithm followed by a digest. If the algorithm is not implemented, unmarshalling of the hashAlg will fail and the TPM will return TPM_RC_HASH.

If the TPM unmarshals the hashAlg of a list entry and the unmarshaled value is not a hash algorithm implemented on the TPM, the TPM shall return TPM_RC_HA.

The pcrHandle parameter is allowed to reference TPM_RH_NULL. If so, the input parameters are processed but no action is taken by the TPM.

NOTE 3 This command allows a list of digests so that PCR in all banks may be updated in a single command. While the semantics of this command allow multiple extends to a single PCR bank, this is not the preferred use and the limit on the number of entries in the list make this use somewhat impractical.
## 24.2.2 Command and Response

### Table 99 — TPM2_PCR_Extend Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PCR_Extend (NV) handle of the PCR</td>
</tr>
<tr>
<td>TPMI_DH_PCR</td>
<td>@pcrHandle</td>
<td>Auth Handle: 1 Auth Role: USER</td>
</tr>
<tr>
<td>TPML_DIGEST_VALUES</td>
<td>digests</td>
<td>list of tagged digest values to be extended</td>
</tr>
</tbody>
</table>

### Table 100 — TPM2_PCR_Extend Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td>.</td>
</tr>
</tbody>
</table>
24.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PCR_Extend_fp.h"

Error Returns | Meaning
--- | ---
TPM_RC_LOCALITY | current command locality is not allowed to extend the PCR referenced by pcrHandle

TPM_RC

TPM2_PCR_Extend(
  PCR_Extend_In     *in  // IN: input parameter list
)
{
  TPM_RC result;
  UINT32 i;

  // Input Validation
  // NOTE: This function assumes that the unmarshaling function for 'digests' will have validated that all of the indicated hash algorithms are valid. If the hash algorithms are correct, the unmarshaling code will unmarshal a digest of the size indicated by the hash algorithm. If the overall size is not consistent, the unmarshaling code will run out of input data or have input data left over. In either case, it will cause an unmarshaling error and this function will not be called.

  // For NULL handle, do nothing and return success
  if (in->pcrHandle == TPM_RH_NULL)
    return TPM_RC_SUCCESS;

  // Check if the extend operation is allowed by the current command locality
  if (!PCRIsExtendAllowed(in->pcrHandle))
    return TPM_RC_LOCALITY;

  // If PCR is state saved and we need to update orderlyState, check NV availability
  if (PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
  {
    result = NvIsAvailable();
    if (result != TPM_RC_SUCCESS) return result;
    g_clearOrderly = TRUE;
  }

  // Internal Data Update
  // Iterate input digest list to extend
  for (i = 0; i < in->digests.count; i++)
  {
    PCRExtend(in->pcrHandle, in->digests.digests[i].hashAlg,
              CryptGetHashDigestSize(in->digests.digests[i].hashAlg),
              (BYTE *) &in->digests.digests[i].digest);
  }

  return TPM_RC_SUCCESS;
}
```
24.3  TPM2_PCR_Event

24.3.1  General Description

This command is used to cause an update to the indicated PCR.

The data in eventData is hashed using the hash algorithm associated with each bank in which the indicated PCR has been allocated. After the data is hashed, the digests list is returned. If the pcrHandle references an implemented PCR and not TPM_ALG_NULL, digests list is processed as in TPM2_PCR_Extend().

A TPM shall support an Event.size of zero through 1,024 inclusive (Event.size is an octet count). An Event.size of zero indicates that there is no data but the indicated operations will still occur,

EXAMPLE 1  If the command implements PCR[2] in a SHA1 bank and a SHA256 bank, then an extend to PCR[2] will cause eventData to be hashed twice, once with SHA1 and once with SHA256. The SHA1 hash of eventData will be Extended to PCR[2] in the SHA1 bank and the SHA256 hash of eventData will be Extended to PCR[2] of the SHA256 bank.

On successful command completion, digests will contain the list of tagged digests of eventData that was computed in preparation for extending the data into the PCR. At the option of the TPM, the list may contain a digest for each bank, or it may only contain a digest for each bank in which pcrHandle is extant.

EXAMPLE 2  Assume a TPM that implements a SHA1 bank and a SHA256 bank and that PCR[22] is only implemented in the SHA1 bank. If pcrHandle references PCR[22], then digests may contain either a SHA1 and a SHA256 digest or just a SHA1 digest.
24.3.2 Command and Response

Table 101 — TPM2_PCR_Event Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PCR_Event (NV)</td>
</tr>
<tr>
<td>TPMI_DH_PCR+</td>
<td>@pcrHandle</td>
<td>Handle of the PCR</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Handle: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_EVENT</td>
<td>eventData</td>
<td>Event data in sized buffer</td>
</tr>
</tbody>
</table>

Table 102 — TPM2_PCR_Event Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPML_DIGEST_VALUES</td>
<td>digests</td>
<td></td>
</tr>
</tbody>
</table>
24.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PCR_Event_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_LOCALITY</td>
<td>current command locality is not allowed to extend the PCR referenced by pcrHandle</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_PCR_Event(
    PCR_Event_In *in, // IN: input parameter list
    PCR_Event_Out *out // OUT: output parameter list
)
{
    TPM_RC result;
    HASH_STATE hashState;
    UINT32 i;
    UINT16 size;

    // Input Validation
    if(in->pcrHandle != TPM_RH_NULL)
    {
        // If the PCR is not allow to extend, return error
        if(!PCRIsExtendAllowed(in->pcrHandle))
            return TPM_RC_LOCALITY;

        // If PCR is state saved and we need to update orderlyState, check NV availability
        if(PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
        {
            result = NvIsAvailable();
            if(result != TPM_RC_SUCCESS) return result;
            g_clearOrderly = TRUE;
        }
    }

    // Internal Data Update
    out->digests.count = HASH_COUNT;

    // Iterate supported PCR bank algorithms to extend
    for(i = 0; i < HASH_COUNT; i++)
    {
        TPM_ALG_ID hash = CryptGetHashAlgByIndex(i);
        out->digests.digests[i].hashAlg = hash;
        size = CryptStartHash(hash, &hashState);
        CryptUpdateDigest2B(&hashState, &in->eventData.b);
        CryptCompleteHash(&hashState, size,
            (BYTE *) &out->digests.digests[i].digest);
        if(in->pcrHandle != TPM_RH_NULL)
            PCRExtend(in->pcrHandle, hash, size,
                (BYTE *) &out->digests.digests[i].digest);
    }
    return TPM_RC_SUCCESS;
}```
24.4  TPM2_PCR_Read

24.4.1  General Description

This command returns the values of all PCR specified in `pcrSelect`. The TPM will process the list of TPMS_PCR_SELECTION in `pcrSelectionIn` in order. Within each TPMS_PCR_SELECTION, the TPM will process the bits in the `pcrSelect` array in ascending PCR order (see Part 2 for definition of the PCR order). If a bit is SET, and the indicated PCR is present, then the TPM will add the digest of the PCR to the list of values to be returned in `pcrValue`.

The TPM will continue processing bits until all have been processed or until `pcrValues` would be too large to fit into the output buffer if additional values were added. The returned `pcrSelectionOut` will have a bit SET in its `pcrSelect` structures for each value present in `pcrValues`.

The current value of the PCR Update Counter is returned in `pcrUpdateCounter`.

The returned list may be empty if none of the selected PCR are implemented.

NOTE  If no PCR are returned from a bank, the selector for the bank will be present in `pcrSelectionOut`.

No authorization is required to read a PCR and any implemented PCR may be read from any locality.
24.4.2 Command and Response

Table 103 — TPM2_PCR_Read Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PCR_Read</td>
</tr>
<tr>
<td>TPML_PCR_SELECTION</td>
<td>pcrSelectionIn</td>
<td>The selection of PCR to read</td>
</tr>
</tbody>
</table>

Table 104 — TPM2_PCR_Read Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>pcrUpdateCounter</td>
<td>the current value of the PCR update counter</td>
</tr>
<tr>
<td>TPML_PCR_SELECTION</td>
<td>pcrSelectionOut</td>
<td>the PCR in the returned list</td>
</tr>
<tr>
<td>TPML_DIGEST</td>
<td>pcrValues</td>
<td>the contents of the PCR indicated in pcrSelect as tagged digests</td>
</tr>
</tbody>
</table>
24.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PCR_Read_fp.h"

TPM_RC
TPM2_PCR_Read(
    PCR_Read_In      *in, // IN: input parameter list
    PCR_Read_Out     *out // OUT: output parameter list
)
{
    // Command Output
    // Call PCR read function. input pcrSelectionIn parameter could be changed
    // to reflect the actual PCR being returned
    PCRRead(&in->pcrSelectionIn, &out->pcrValues, &out->pcrUpdateCounter);
    out->pcrSelectionOut = in->pcrSelectionIn;
    return TPM_RC_SUCCESS;
}
```
24.5 TPM2_PCR_Allocate

24.5.1 General Description

This command is used to set the desired PCR allocation of PCR and algorithms. This command requires platformAuth.

The TPM will evaluate the request and, if sufficient memory is available for the requested allocation, the TPM will store the allocation request for use during the next TPM2_Startup(TPM_SU_CLEAR) operation. The PCR allocation in place when this command is executed will be retained until the next TPM2_Startup(TPM_SU_CLEAR).

If no allocation is specified for a bank, then no PCR will be allocated to that bank. If a bank is listed more than once, then the last selection in the pcrAllocation list is the one that the TPM will attempt to allocate.

This command shall not allocate more PCR in any bank than there are PCR attribute definitions. The PCR attribute definitions indicate how a PCR is to be managed – if it is resettable, the locality for update, etc. In the response to this command, the TPM returns the maximum number of PCR allowed for any bank.

If the command is properly authorized, it will return SUCCESS even though the request fails. This is to allow the TPM to return information about the size needed for the requested allocation and the size available. If the sizeNeeded parameter in the return is less than or equal to the sizeAvailable parameter, then the allocationSuccess parameter will be YES.

After this command, TPM2_Shutdown() is only allowed to have a startupType equal to TPM_SU_CLEAR.

NOTE Even if this command does not cause the PCR allocation to change, the TPM cannot have its state saved. This is done in order to simplify the implementation. There is no need to optimize this command as it is not expected to be used more than once in the lifetime of the TPM (it can be used any number of times but there is no justification for optimization).
### 24.5.2 Command and Response

#### Table 105 — TPM2_PCR_Allocate Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PCR_Allocate {NV}</td>
</tr>
<tr>
<td>TPMI_RH_PLATFORM</td>
<td>@authHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPML_PCR_SELECTION</td>
<td>pcrAllocation</td>
<td>the requested allocation</td>
</tr>
</tbody>
</table>

#### Table 106 — TPM2_PCR_Allocate Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMI_YES_NO</td>
<td>allocationSuccess</td>
<td>YES if the allocation succeeded</td>
</tr>
<tr>
<td>UINT32</td>
<td>maxPCR</td>
<td>maximum number of PCR that may be in a bank</td>
</tr>
<tr>
<td>UINT32</td>
<td>sizeNeeded</td>
<td>number of octets required to satisfy the request</td>
</tr>
<tr>
<td>UINT32</td>
<td>sizeAvailable</td>
<td>Number of octets available. Computed before the allocation.</td>
</tr>
</tbody>
</table>
24.5.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PCR_Allocate_fp.h"

TPM_RC
TPM2_PCR_Allocate(
    PCR_Allocate_In     *in,          // IN: input parameter list
    PCR_Allocate_Out    *out         // OUT: output parameter list
)
{
    TPM_RC      result;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point.
    // Note: These codes are not listed in the return values above because it is
    // an implementation choice to check in this routine rather than in a common
    // function that is called before these actions are called. These return values
    // are described in the Response Code section of Part 3.
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

    // Command Output

    // Call PCR Allocation function.
    out->allocationSuccess = PCRAllocate(&in->pcrAllocation, &out->maxPCR,
                                          &out->sizeNeeded, &out->sizeAvailable);

    // if re-configuration succeeds, set the flag to indicate PCR configuration is
    // going to be changed in next boot
    if(out->allocationSuccess == YES)
        g_pcrReConfig = TRUE;

    return TPM_RC_SUCCESS;
} 
```
24.6  TPM2_PCR_SetAuthPolicy

24.6.1 General Description

This command is used to associate a policy with a PCR or group of PCRs. The policy determines the conditions under which a PCR may be extended or reset.

A policy may only be associated with a PCR that has been defined by a platform-specific specification as allowing a policy. If the TPM implementation does not allow a policy for pcrNum, the TPM shall return TPM_RC_VALUE.

A platform-specific specification may group PCRs so that they share a common policy. In such case, a pcrNum that selects any of the PCRs in the group will change the policy for all PCRs in the group.

The policy setting is persistent and may only be changed by TPM2_PCR_SetAuthPolicy() or by TPM2_ChangePPS().

Before this command is first executed on a TPM or after TPM2_ChangePPS(), the access control on the PCR will be set to the default value defined in the platform-specific specification.

NOTE 1  It is expected that the typical default will be with the policy hash set to TPM_ALG_NULL and an Empty Buffer for the authPolicy value. This will allow an EmptyAuth to be used as the authorization value.

If the size of the data buffer in authPolicy is not the size of a digest produced by hashAlg, the TPM shall return TPM_RC_SIZE.

NOTE 2  If hashAlg is TPM_ALG_NULL, then the size is required to be zero.

This command requires platformAuth/platformPolicy.

NOTE 3  If the PCR is in multiple policy sets, the policy will be changed in only one set. The set that is changed will be implementation dependent.
### 24.6.2 Command and Response

#### Table 107 — TPM2_PCR_SetAuthPolicy Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PCR_SetAuthPolicy (NV)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPM_RH_PLATFORM+{PP}</td>
</tr>
<tr>
<td>TPMI_RH_PLATFORM</td>
<td>@authHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>authPolicy</td>
<td>the desired authPolicy</td>
</tr>
<tr>
<td>TPMI_ALG_HASH+</td>
<td>policyDigest</td>
<td>the digest of the policy</td>
</tr>
<tr>
<td>TPMI_DH_PCR</td>
<td>pcrNum</td>
<td>the PCR for which the policy is to be set</td>
</tr>
</tbody>
</table>

#### Table 108 — TPM2_PCR_SetAuthPolicy Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
24.6.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PCR_SetAuthPolicy_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIZE</td>
<td>size of authPolicy is not the size of a digest produced by policyDigest</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>PCR referenced by pcrNum is not a member of a PCR policy group</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_PCR_SetAuthPolicy(
    PCR_SetAuthPolicy_In     *in // IN: input parameter list
)
{
    UINT32      groupIndex;
    TPM_RC      result;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Input Validation:
    // Check the authPolicy consistent with hash algorithm
    if(in->authPolicy.t.size != CryptGetHashDigestSize(in->policyDigest))
        return TPM_RC_SIZE + RC_PCR_SetAuthPolicy_authPolicy;

    // If PCR does not belong to a policy group, return TPM_RC_VALUE
    if(!PCRBelongsPolicyGroup(in->pcrNum, &groupIndex))
        return TPM_RC_VALUE + RC_PCR_SetAuthPolicy_pcrNum;

    // Internal Data Update
    // Set PCR policy
    gp.pcrPolicies.hashAlg[groupIndex] = in->policyDigest;
    gp.pcrPolicies.policy[groupIndex] = in->authPolicy;

    // Save new policy to NV
    NvWriteReserved(NV_PCR_POLICIES, &gp.pcrPolicies);

    return TPM_RC_SUCCESS;
}
```
24.7  TPM2_PCR_SetAuthValue

24.7.1  General Description

This command changes the authValue of a PCR or group of PCR.

An authValue may only be associated with a PCR that has been defined by a platform-specific specification as allowing an authorization value. If the TPM implementation does not allow an authorization for pcrNum, the TPM shall return TPM_RC_VALUE. A platform-specific specification may group PCR so that they share a common authorization value. In such case, a pcrNum that selects any of the PCR in the group will change the authValue value for all PCR in the group.

The authorization setting is set to EmptyAuth on each STARTUP(CLEAR) or by TPM2_Clear(). The authorization setting is preserved by SHUTDOWN(STATE).
## 24.7.2 Command and Response

### Table 109 — TPM2_PCR_SetAuthValue Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PCR_SetAuthValue handle for a PCR that may have an authorization value set</td>
</tr>
<tr>
<td>TPMI_DH_PCR</td>
<td>@pcrHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>auth</td>
<td>the desired authorization value</td>
</tr>
</tbody>
</table>

### Table 110 — TPM2_PCR_SetAuthValue Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
24.7.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PCR_SetAuthValue_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>PCR referenced by pcrHandle is not a member of a PCR authorization group</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_PCR_SetAuthValue(
    PCR_SetAuthValue_In *in       // IN: input parameter list
) {
    UINT32 groupIndex;
    TPM_RC result;

    // Input Validation:
    if(!PCRBelongsToAuthGroup(in->pcrHandle, &groupIndex))
        return TPM_RC_VALUE;

    // The command may cause the orderlyState to be cleared due to the update of
    // state clear data. If this is the case, check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    if(gp.orderlyState != SHUTDOWN_NONE)
        { result = NvIsAvailable();
            if(result != TPM_RC_SUCCESS) return result;
            g_clearOrderly = TRUE;
        }

    // Internal Data Update
    // Set PCR authValue
    gc.pcrAuthValues.auth[groupIndex] = in->auth;
    return TPM_RC_SUCCESS;

```
24.8 TPM2_PCR_Reset

24.8.1 General Description

If the attribute of a PCR allows the PCR to be reset and proper authorization is provided, then this command may be used to set the PCR to zero. The attributes of the PCR may restrict the locality that can perform the reset operation.

NOTE 1 The definition of TPMI_DH_PCR in Part 2 indicates that if pcrHandle is out of the allowed range for PCR, then the appropriate return value is TPM_RC_VALUE.

If pcrHandle references a PCR that cannot be reset, the TPM shall return TPM_RC_LOCALITY.

NOTE 2 TPM_RC_LOCALITY is returned because the reset attributes are defined on a per-locality basis.
### 24.8.2 Command and Response

<table>
<thead>
<tr>
<th>Table 111 — TPM2_PCR_Reset Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_CC</td>
</tr>
<tr>
<td>TPMI_DH_PCR</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 112 — TPM2_PCR_Reset Response</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TPM_ST</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_RC</td>
</tr>
</tbody>
</table>
24.8.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PCR_Reset_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_LOCALITY</td>
<td>current command locality is not allowed to reset the PCR referenced by pcrHandle</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_PCR_Reset(
    PCR_Reset_In *in // IN: input parameter list
)
{
    TPM_RC result;

    // Input Validation
    if (!PCRIsResetAllowed(in->pcrHandle))
        return TPM_RC_LOCALITY;

    // If PCR is state saved and we need to update orderlyState, check NV availability
    if (PCRIsStateSaved(in->pcrHandle) && gp.orderlyState != SHUTDOWN_NONE)
        { 
            result = NvIsAvailable();
            if (result != TPM_RC_SUCCESS)
                return result;
            g_clearOrderly = TRUE;
        }

    // Internal Data Update
    PCRSetValue(in->pcrHandle, 0);

    // Indicate that the PCR changed so that pcrCounter will be incremented if necessary.
    PCRChanged(in->pcrHandle);
    return TPM_RC_SUCCESS;
}
```
24.9 _TPM_Hash_Start

24.9.1 Description

This indication from the TPM interface indicates the start of a dynamic Core Root of Trust for Measurement (D-CRTM) measurement sequence. On receipt of this indication, the TPM will initialize an Event sequence context.

If no object memory is available for creation of the sequence context, the TPM will flush the context of an object so that creation of the Event sequence context will always succeed.

A platform-specific specification may allow this indication before TPM2_Startup().

NOTE If this indication occurs after TPM2_Startup(), it is the responsibility of software to ensure that an object context slot is available or to deal with the consequences of having the TPM select an arbitrary object to be flushed. If this indication occurs before TPM2_Startup() then all context slots are available.
#include "InternalRoutines.h"

This function is called to process a _TPM_Hash_Start() indication.

```c
void _TPM_Hash_Start(void)
{
    TPM_RC result;
    TPMI_DH_OBJECT handle;

    // If a DRTM sequence object exists, terminate it.
    if(g_DRTMHandle != TPM_RH_UNASSIGNED)
    {
        ObjectTerminateEvent();
    }

    // Create an event sequence object and store the handle in global
    // g_DRTMHandle. A TPM_RC_OBJECT_MEMORY error may be returned at this point
    // The null value for the 'auth' parameter will cause the sequence structure to
    // be allocated without being set as present. This keeps the sequence from
    // being left behind if the sequence is terminated early.
    result = ObjectCreateEventSequence(NULL, &g_DRTMHandle);

    // If a free slot was not available, then free up a slot.
    if(result != TPM_RC_SUCCESS)
    {
        // An implementation does not need to have a fixed relationship between
        // slot numbers and handle numbers. To handle the general case, scan for
        // a handle that is assigned an free it for the DRTM sequence.
        // In the reference implementation, the relationship between handles and
        // slots is fixed. So, if the call to ObjectCreateEvenSequence()
        // failed indicating that all slots are occupied, then the first handle we
        // are going to check (TRANSIENT_FIRST) will be occupied. It will be freed
        // so that it can be assigned for use as the DRTM sequence object.
        for(handle = TRANSIENT_FIRST; handle < TRANSIENT_LAST; handle++)
        {
            // try to flush the first object
            if(ObjectIsPresent(handle))
            {
                break;
            }
        }

        // If the first call to find a slot fails but none of the slots is occupied
        // then there's a big problem
        pAssert(handle < TRANSIENT_LAST);
    }

    // Free the slot
    ObjectFlush(handle);

    // Try to create an event sequence object again. This time, we must
    // succeed.
    result = ObjectCreateEventSequence(NULL, &g_DRTMHandle);
    pAssert(result == TPM_RC_SUCCESS);
}
```

return;
```
24.10 _TPM_Hash_Data

24.10.1 Description

This indication from the TPM interface indicates arrival of one or more octets of data that are to be included in the Core Root of Trust for Measurement (CRTM) sequence context created by the _TPM_Hash_Start indication. The context holds data for each hash algorithm for each PCR bank implemented on the TPM.

If no DRTM Event Sequence context exists, this indication is discarded and no other action is performed.
24.10.2 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Platform.h"

This function is called to process a _TPM_Hash_Data() indication.

void _TPM_Hash_Data(
    UINT32       dataSize, // IN: size of data to be extend
    BYTE         *data   // IN: data buffer
)
{
    UINT32    i;
    HASH_OBJECT *hashObject;

    if (g_DRTMHandle == TPM_RH_UNASSIGNED)
        return;

    hashObject = (HASH_OBJECT *)ObjectGet(g_DRTMHandle);
    pAssert(hashObject->attributes.eventSeq);

    for(i = 0; i < HASH_COUNT; i++)
        CryptUpdateDigest(&hashObject->state.hashState[i], dataSize, data);

    return;
}
```
24.11 _TPM_Hash_End

24.11.1 Description

This indication from the TPM interface indicates the end of the CRTM measurement. This indication is discarded and no other action performed if the TPM does not contain a CRTM Event sequence context.

NOTE A CRTM Event Sequence context is created by _TPM_Hash_Start().

If the CRTM Event sequence occurs after TPM2_Startup(), the TPM will set all of the PCR designated in the platform-specific specifications as resettable by this event to the value indicated in the platform specific specification, and increment restartCount. The TPM will then Extend the Event Sequence digest/digests into the designated, DRTM PCR.

\[
\text{PCR[DRTM]}[\text{hashAlg}] := H_{\text{hashAlg}}(\text{initial\_value} || H_{\text{hashAlg}}(\text{hash\_data}))
\]  

where

- DRTM: index for CRTM PCR designated by a platform-specific specification
- hashAlg: hash algorithm associated with a bank of PCR
- initial_value: initialization value specified in the platform-specific specification (should be 0…0)
- hash_data: all the octets of data received in _TPM_Hash_Data indications

A _TPM_Hash_End indication that occurs after TPM2_Startup() will increment pcrUpdateCounter unless a platform-specific specification excludes modifications of PCR[DRTM] from causing an increment.

A platform-specific specification may allow an H-CRTM Event Sequence before TPM2_Startup(). If so, _TPM_Hash_End will complete the digest, initialize PCR[0] with a digest-size value of 4, and then extend the H-CRTM Event Sequence data into PCR[0].

\[
\text{PCR[0]}[\text{hashAlg}] := H_{\text{hashAlg}}(0…04 || H_{\text{hashAlg}}(\text{hash\_data}))
\]

NOTE The entire sequence of _TPM_Hash_Start, _TPM_Hash_Data, and _TPM_Hash_End are required to complete before TPM2_Startup() or the sequence will have no effect on the TPM.
#include "InternalRoutines.h"

This function is called to process a _TPM_Hash_End() indication.

```c
void _TPM_Hash_End(void) {
    UINT32 i;
    TPM2B_DIGEST digest;
    HASH_OBJECT *hashObject;
    TPM1_DH_PCR pcrHandle;

    // If the DRTM handle is not being used, then either _TPM_Hash_Start has not
    // been called, _TPM_Hash_End was previously called, or some other command
    // was executed and the sequence was aborted.
    if (g_DRTMHandle == TPM_RH_UNASSIGNED)
        return;

    // Get DRTM sequence object
    hashObject = (HASH_OBJECT *)ObjectGet(g_DRTMHandle);

    // Is this _TPM_Hash_End after Startup or before
    if (TPMIsStarted())
        {
        // After
        // Reset the DRTM PCR
        PCRResetDynamics();
        // Extend the DRTM PCR.
        pcrHandle = PCR_FIRST + DRTM_PCR;
        // DRTM sequence increments restartCount
        gr.restartCount++;
        }
    else
        {
        pcrHandle = PCR_FIRST + HCRM_PCR;
        }

    // Complete hash and extend PCR, or if this is an HCRM, complete
    // the hash and write the PCR
    for (i = 0; i < HASH_COUNT; i++)
        {
        TPM1_ALG_HASH hash = CryptGetHashAlgByIndex(i);
        // Complete hash
digest.t.size = CryptGetHashDigestSize(hash);
        CryptCompleteHash2B(hashObject->state.hashState[i], &digest.b);
        // If this is DRTM, extend to zeroed PCR
        // If this is H-DRTM, copy to HCRM PCR
        if (TPMIsStarted())
            // Extend PCR
            PCRExtend(pcrHandle, hash, digest.t.size, digest.t.buffer);
        else
            PcrWrite(pcrHandle, hash, &digest);
        }
    ```
58     }
59
60     // Flush sequence object.
61     ObjectFlush(g_DRTMHandle);
62
63     g_DRTMHandle = TPM_RH_UNASSIGNED;
64
65     g_DrtmPreStartup = TRUE;
66
67     return;
68 }
25 Enhanced Authorization (EA) Commands

25.1 Introduction

The commands in this clause are used for policy evaluation. When successful, each command will update the $policySession→policyDigest$ in a policy session context in order to establish that the authorizations required to use an object have been provided. Many of the commands will also modify other parts of a policy context so that the caller may constrain the scope of the authorization that is provided.

NOTE 1 Many of the terms used in this clause are described in detail in Part 1 and are not redefined in this clause.

The $policySession$ parameter of the command is the handle of the policy session context to be modified by the command.

If the $policySession$ parameter indicates a trial policy session, then the $policySession→policyDigest$ will be updated and the indicated validations are not performed.

NOTE 2 A policy session is a trial policy by $TPM2_StartAuthSession(sessionType = TPM_SE_TRIAL)$.

NOTE 3 Unless there is an unmarshaling error in the parameters of the command, these commands will return $TPM_RC_SUCCESS$ when $policySession$ references a trial session.

NOTE 4 Policy context other than the $policySession→policyDigest$ may be updated for a trial policy but it is not required.
25.2 Signed Authorization Actions

25.2.1 Introduction

The TPM2_PolicySigned, TPM_PolicySecret, and TPM2_PolicyTicket commands use many of the same functions. This clause consolidates those functions to simplify the document and to ensure uniformity of the operations.

25.2.2 Policy Parameter Checks

These parameter checks will be performed when indicated in the description of each of the commands:

a) nonceTPM – If this parameter is not the Empty Buffer, and it does not match policySession→nonceTPM, then the TPM shall return TPM_RC_VALUE.

b) expiration – If this parameter is not zero, then its absolute value is compared to the time in seconds since the policySession→nonceTPM was generated. If more time has passed than indicted in expiration, the TPM shall return TPM_RC_EXPIRED. If nonceTPM is the Empty buffer, and expiration is non-zero, then the TPM shall return TPM_RC_EXPIRED.

c) timeout – This parameter is compared to the current TPM time. If policySession→timeout is in the past, then the TPM shall return TPM_RC_EXPIRED.

NOTE 1 The expiration parameter is present in the TPM2_PolicySigned and TPM2_PolicySecret command and timeout is the analogous parameter in the TPM2_PolicyTicket command.

d) cpHashA – If this parameter is not an Empty Buffer

NOTE 2 CpHashA is the hash of the command to be executed using this policy session in the authorization. The algorithm used to compute this hash is required to be the algorithm of the policy session.

1) the TPM shall return TPM_RC_CPHASH if policySession→cpHash does not have its default value or the contents of policySession→cpHash are not the same as cpHashA; or

NOTE 3 CpHash is the expected cpHash value held in the policy session context.

2) the TPM shall return TPM_RC_SIZE if cpHashA is not the same size as policySession→policyDigest.

NOTE 4 PolicySession→policyDigest is the size of the digest produced by the hash algorithm used to compute policyDigest.
25.2.3 **PolicyDigest Update Function (PolicyUpdate())**

This is the update process for policySession→policyDigest used by TPM2_PolicySigned(), TPM2_PolicySecret(), TPM2_PolicyTicket(), and TPM2_PolicyAuthorize(). The function prototype for the update function is:

\[
\text{PolicyUpdate}(\text{commandCode, arg2, arg3})
\]

where

\[
\begin{align*}
\text{arg2} & : \text{a TPM2B\_NAME} \\
\text{arg3} & : \text{a TPM2B}
\end{align*}
\]

These parameters are used to update policySession→policyDigest by

\[
policyDigest_{\text{new}} := H_{\text{policyAlg}}(policyDigest_{\text{old}} || \text{commandCode} || \text{arg2.name})
\]

followed by

\[
policyDigest_{\text{new+1}} := H_{\text{policyAlg}}(policyDigest_{\text{new}} || \text{arg3.buffer})
\]

where

\[
H_{\text{policyAlg}}() \quad \text{the hash algorithm chosen when the policy session was started}
\]

**NOTE 1** If arg3 is a TPM2B\_NAME, then arg3.buffer will actually be an arg3.name.

**NOTE 2** The arg2.size and arg3.size fields are not included in the hashes.

**NOTE 3** PolicyUpdate() uses two hashes because arg2 and arg3 are variable-sized and the concatenation of arg2 and arg3 in a single hash could produce the same digest even though arg2 and arg3 are different. Processing of the arguments separately in different Extend operation insures that the digest produced by PolicyUpdate() will be different if arg2 and arg3 are different.
25.2.4 Policy Context Updates

When a policy command modifies some part of the policy session context other than the policySession→policyDigest, the following rules apply.

- **cpHash** – this parameter may only be changed if it contains its initialization value (an Empty String). If cpHash is not the Empty String when a policy command attempts to update it, the TPM will return an error (TPM_RC_CPHASH) if the current and update values are not the same.

- **timeOut** – this parameter may only be changed to a smaller value. If a command attempts to update this value with a larger value (longer into the future), the TPM will discard the update value. This is not an error condition.

- **commandCode** – once set by a policy command, this value may not be change except by TPM2_PolicyRestart(). If a policy command tries to change this to a different value, an error is returned (TPM_RC_POLICY_CC).

- **pcrUpdateCounter** – this parameter is updated by TPM2_PolicyPCR(). This value may only be set once during a policy. Each time TPM2_PolicyPCR() executes, it checks to see if policySession→pcrUpdateCounter has its default state indicating that this is the first TPM2_PolicyPCR(). If it has its default value, then policySession→pcrUpdateCounter is set to the current value of pcrUpdateCounter. If policySession→pcrUpdateCounter does not have its default value and its value is not the same as pcrUpdateCounter, the TPM shall return TPM_RC_PCR_CHANGED.

**NOTE**

If this parameter and pcrUpdateCounter are not the same, it indicates that PCR have changed since checked by the previous TPM2_PolicyPCR(). Since they have changed, the previous PCR validation is no longer valid.

- **commandLocality** – this parameter is the logical AND of all enabled localities. All localities are enabled for a policy when the policy session is created. TPM2_PolicyLocalities() selectively disables localities. Once use of a policy for a locality has been disabled, it cannot be enabled except by TPM2_PolicyRestart().

- **isPPRequired** – once SET, this parameter may only be CLEARed by TPM2_PolicyRestart().

- **isAuthValueNeeded** – once SET, this parameter may only be CLEARed by TPM2_PolicyPassword() or TPM2_PolicyRestart().

- **isPasswordNeeded** – once SET, this parameter may only be CLEARed by TPM2_PolicyAuthValue() or TPM2_PolicyRestart().

**NOTE**

Both TPM2_PolicyAuthValue() and TPM2_PolicyPassword() change policySession→policyDigest in the same way. The different commands simply indicate to the TPM the format used for the authValue (HMAC or clear text). Both commands could be in the same policy. The final instance of these commands determines the format.
25.2.5 Policy Ticket Creation

If for TPM2_PolicySigned() or TPM2_PolicySecret() the caller specified a negative value for expiration, and the policy update succeeds, then the TPM will return a ticket that includes a value indicating when the authorization expires. The required computation for the digest in the authorization ticket is:

\[
\text{HMAC}(proof, H_{\text{policyAlg}}(ticketType || timeout || cpHashA || policyRef || authObject\rightarrow\text{Name}))
\]

where

- **proof** secret associated with the storage primary seed (SPS) of the TPM
- \(H_{\text{policyAlg}}\) hash function using the hash algorithm associated with the policy session
- **ticketType** either TPM_ST_AUTH_SECRET or TPM_ST_AUTH_SIGNED, used to indicate type of the ticket

**NOTE 1** If the ticket is produced by TPM2_PolicySecret() then ticketType is TPM_ST_AUTH_SECRET and if produced by TPM2_PolicySigned() then ticketType is TPM_ST_AUTH_SIGNED.

- **timeout** implementation-specific representation of the expiration time of the ticket; required to be the implementation equivalent of \(policySession\rightarrow\text{startTime}\) plus the absolute value of expiration

**NOTE 2** Timeout is not the same as expiration. The expiration value in the aHash is a relative time, using the creation time of the authorization session (TPM2_StartAuthSession()) as its reference. The timeout parameter is an absolute time, using TPM Clock as the reference.

- **cpHashA** the command parameter digest for the command being authorized; computed using the hash algorithm of the policy session
- **policyRef** the commands that use this function have a policyRef parameter and the value of that parameter is used here
- **authObject\rightarrow\text{Name}** Name associated with the authObject parameter
25.3 TPM2_PolicySigned

25.3.1 General Description

This command includes a signed authorization in a policy. The command ties the policy to a signing key by including the Name of the signing key in the policyDigest.

If policySession is a trial session, the TPM will not check the signature and will update policySession→policyDigest as described in 25.2.3 as if a properly signed authorization was received; but no ticket will be produced.

If policySession is not a trial session, the TPM will validate auth and only perform the update if it is a valid signature over the fields of the command.

The authorizing object will sign a digest of the authorization qualifiers: nonceTPM, expiration, cpHashA, and policyRef. The digest is computed as:

\[
aHash := H_{\text{authAlg}}(\text{nonceTPM} || \text{expiration} || \text{cpHashA} || \text{policyRef})
\]  

where

\[
H_{\text{authAlg}}() \]  

the hash associated with the auth parameter of this command

NOTE 1 Each signature and key combination indicates the scheme and each scheme has an associated hash.

nonceTPM the nonceTPM parameter from the TPM2_StartAuthSession() response. If the authorization is not limited to this session, the size of this value is zero

expiration time limit on authorization set by authorizing object. This 32-bit value is set to zero if the expiration time is not being set.

cpHashA digest of the command parameters for the command being approved using the hash algorithm of the policy session. Set to an EmptyAuth if the authorization is not limited to a specific command.

NOTE 2 This is not the cpHash of this TPM2_PolicySigned() command.

policyRef an opaque value determined by the authorizing entity. Set to the Empty Buffer if no value is present.

EXAMPLE The computation for an aHash if there are no restrictions is:

\[
aHash := H_{\text{authAlg}}(00 00 00 00_{16})
\]

which is the hash of an expiration time of zero.

The aHash is signed by the private key associated with key. The signature and signing parameters are combined to create the auth parameter.

The TPM will perform the parameter checks listed in 25.2.2.

If the parameter checks succeed, the TPM will construct a test digest (tHash) over the provided parameters using the same formulation a shown in equation (13) above.

If tHash does not match the digest of the signed aHash, then the authorization fails and the TPM shall return TPM_RC_POLICY_FAIL and make no change to policySession→policyDigest.
When all validations have succeeded, \( policySession \rightarrow policyDigest \) is updated by \textbf{PolicyUpdate()} (see 25.2.3).

\begin{align*}
\textbf{PolicyUpdate}(TPM\_CC\_PolicySigned, authObject \rightarrow Name, policyRef)
\end{align*}

(14)

If the \( cpHashA \) parameter is not an Empty Buffer, it is copied to \( policySession \rightarrow cpHash \).

The TPM will optionally produce a ticket as described in 25.2.5.

Authorization to use \textit{authObject} is not required.
### 25.3.2 Command and Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicySigned handle for a public key that will validate the signature Auth Index: None</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>authObject</td>
<td>handle for the policy session being extended Auth Index: None</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td></td>
</tr>
<tr>
<td>TPM2B_NONCE</td>
<td>nonceTPM</td>
<td>the policy nonce for the session If the nonce is not included in the authorization qualification, this field is the Empty Buffer.</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>cpHashA</td>
<td>digest of the command parameters to which this authorization is limited This is not the cpHash for this command but the cpHash for the command to which this policy session will be applied. If it is not limited, the parameter will be the Empty Buffer.</td>
</tr>
<tr>
<td>TPM2B_NONCE</td>
<td>policyRef</td>
<td>a reference to a policy relating to the authorization – may be the Empty Buffer Size is limited to be no larger than the nonce size supported on the TPM.</td>
</tr>
<tr>
<td>INT32</td>
<td>expiration</td>
<td>time when authorization will expire, measured in seconds from the time that nonceTPM was generated. If expiration is zero, a NULL Ticket is returned.</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>auth</td>
<td>signed authorization (not optional)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_TIMEOUT</td>
<td>timeout</td>
<td>implementation-specific time value, used to indicate to the TPM when the ticket expires NOTE If policyTicket is a NULL Ticket, then this shall be the Empty Buffer.</td>
</tr>
<tr>
<td>TPMT_TK_AUTH</td>
<td>policyTicket</td>
<td>produced if the command succeeds and expiration in the command was non-zero; this ticket will use the TPMT_ST_AUTH_SIGNED structure tag</td>
</tr>
</tbody>
</table>
25.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Policy_spt_fp.h"
#include "PolicySigned_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CPHASH</td>
<td>cpHash was previously set to a different value</td>
</tr>
<tr>
<td>TPM_RC_EXPIRED</td>
<td>expiration indicates a time in the past or expiration is non-zero but no</td>
</tr>
<tr>
<td>TPM_RC_HANDLE</td>
<td>authObject need to have sensitive portion loaded</td>
</tr>
<tr>
<td>TPM_RC_KEY</td>
<td>authObject is not a signing scheme</td>
</tr>
<tr>
<td>TPM_RC_NONCE</td>
<td>nonceTPM is not the nonce associated with the policySession</td>
</tr>
<tr>
<td>TPM_RC_SCHEME</td>
<td>the signing scheme of auth is not supported by the TPM</td>
</tr>
<tr>
<td>TPM_RC_SIGNATURE</td>
<td>the signature is not genuine</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>input cpHash has wrong size</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>input policyID or expiration does not match the internal data in policy</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_PolicySigned(
    PolicySigned_In *in, // IN: input parameter list
    PolicySigned_Out *out // OUT: output parameter list
)
{
    TPM_RC result = TPM_RC_SUCCESS;
    SESSION *session;
    OBJECT *authObject;
    TPM2B_NAME entityName;
    TPM2B_DIGEST authHash;
    HASH_STATE hashState;
    UINT32 expiration = (in->expiration < 0)
    ? -(in->expiration) : in->expiration;
    UINT64 authTimeout = 0;

    // Input Validation
    session = SessionGet(in->policySession); // the session structure
    authObject = ObjectGet(in->authObject);  // pointer for the object
    // providing authorization
    // signature

    // Only do input validation if this is not a trial policy session
    if(session->attributes.isTrialPolicy == CLEAR)
    {
        if(expiration != 0)
            authTimeout = expiration * 1000 + session->startTime;

        result = PolicyParameterChecks(session, authTimeout,
            &in->cpHashA, &in->nonceTPM,
            RC_PolicySigned_nonceTPM,
            RC_PolicySigned_cpHashA,
            RC_PolicySigned_expiration);

        if(result != TPM_RC_SUCCESS)
```
Part 3: Commands

```
return result;

// Re-compute the digest being signed
*(See part 3 specification)
// The digest is computed as:
// aHash := hash (nonceTPM | expiration | cpHashA | policyRef)
// where:
// hash() the hash associated with the signed auth
// nonceTPM the nonceTPM value from the TPM2_StartAuthSession.
// response If the authorization is not limited to this
// session, the size of this value is zero.
// expiration time limit on authorization set by authorizing object.
// This 32-bit value is set to zero if the expiration
// time is not being set.
// cpHashA hash of the command parameters for the command being
// approved using the hash algorithm of the PSAP session.
// Set to NULLauth if the authorization is not limited
// to a specific command.
// policyRef hash of an opaque value determined by the authorizing
// object. Set to the NULLdigest if no hash is present.
*/

// Start hash
authHash.t.size = CryptStartHash(GetSignHashAlg(&in->auth),
&hashState);

// add nonceTPM
CryptUpdateDigest2B(&hashState, &in->nonceTPM.b);

// add expiration
CryptUpdateDigestInt(&hashState, sizeof(UINT32), (BYTE*) &in->expiration);

// add cpHashA
CryptUpdateDigest2B(&hashState, &in->cpHashA.b);

// add policyRef
CryptUpdateDigest2B(&hashState, &in->policyRef.b);

// Complete digest
CryptCompleteHash2B(&hashState, &authHash.b);

// Validate Signature. A TPM_RC_SCHEME, TPM_RC_TYPE or TPM_RC_SIGNATURE
// error may be returned at this point
result = CryptVerifySignature(in->authObject, &authHash, &in->auth);
if(result != TPM_RC_SUCCESS)
  return RcSafeAddToResult(result, RC_PolicySigned_auth);
}

// Internal Data Update
// Need the Name of the signing entity
entityName.t.size = EntityGetName(in->authObject, &entityName.t.name);

// Update policy with input policyRef and name of auth key
// These values are updated even if the session is a trial session
PolicyContextUpdate(TPM_CC_PolicySigned, &entityName, &in->policyRef,
&in->cpHashA, authTimeout, session);

// Command Output

// Create ticket and timeout buffer if in->expiration < 0 and this is not
// a trial session.
// NOTE: PolicyParameterChecks() makes sure that nonceTPM is present
// when expiration is non-zero.
if(in->expiration < 0
  && session->attributes.isTrialPolicy == CLEAR
)
{ // Generate timeout buffer. The format of output timeout buffer is
  // TPM-specific.
  // Note: can't do a direct copy because the output buffer is a byte
  // array and it may not be aligned to accept a 64-bit value. The method
  // used has the side-effect of making the returned value a big-endian,
  // 64-bit value that is byte aligned.
  out->timeout.t.size = sizeof(UINT64);
  UINT64_TO_BYTE_ARRAY(authTimeout, out->timeout.t.buffer);

  // Compute policy ticket
  TicketComputeAuth(TPM_ST_AUTH_SIGNED, EntityGetHierarchy(in->authObject),
                    authTimeout, &in->cpHashA, &in->policyRef, &entityName,
                    &out->policyTicket);
} else {
  // Generate a null ticket.
  // timeout buffer is null
  out->timeout.t.size = 0;

  // auth ticket is null
  out->policyTicket.tag = TPM_ST_AUTH_SIGNED;
  out->policyTicket.hierarchy = TPM_RH_NULL;
  out->policyTicket.digest.t.size = 0;
}

return TPM_RC_SUCCESS;
25.4 TPM2_PolicySecret

25.4.1 General Description

This command includes a secret-based authorization to a policy. The caller proves knowledge of the secret value using an authorization session using the authValue associated with authHandle. A password session, an HMAC session, or a policy session containing TPM2_PolicyAuthValue() or TPM2_PolicyPassword() will satisfy this requirement.

“If a policy session is used and use of the authValue of authHandle is not required, the TPM will return TPM_RC_MODE.”

The secret is the authValue of authObject, which may be any TPM entity with a handle and an associated authValue. This includes the reserved handles (for example, Platform, Storage, and Endorsement), NV Indexes, and loaded objects.

NOTE 1 The authorization value for a hierarchy cannot be used in this command if the hierarchy is disabled.

If the authorization check fails, then the normal dictionary attack logic is invoked.

If the authorization provided by the authorization session is valid, the command parameters are checked as described in 25.2.2.

When all validations have succeeded, policySession→policyDigest is updated by PolicyUpdate() (see 25.2.3).

\[
\text{PolicyUpdate}(\text{TPM_CC_PolicySecret, authObject→Name, policyRef})
\]

(15)

If the cpHashA command parameter is not an Empty Buffer, it is copied to cpHash in the session context.

The TPM will optionally produce a ticket as described in 25.2.5.

If the session is a trial session, policySession→policyDigest is updated as if the authorization is valid but no check is performed.

NOTE 2 If an HMAC is used to convey the authorization, a separate session is needed for the authorization. Because the HMAC in that authorization will include a nonce that prevents replay of the authorization, the value of the nonceTPM parameter in this command is limited. It is retained mostly to provide processing consistency with TPM2_PolicySigned().
### 25.4.2 Command and Response

#### Table 115 — TPM2_PolicySecret Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicySecret handle for an entity providing the authorization</td>
</tr>
<tr>
<td>TPMI_DH_ENTITY</td>
<td>@authHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_NONCE</td>
<td>nonceTPM</td>
<td>the policy nonce for the session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the nonce is not included in the authorization qualification, this field is the Empty Buffer.</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>cpHashA</td>
<td>digest of the command parameters to which this authorization is limited</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This not the cpHash for this command but the cpHash for the command to which this policy session will be applied. If it is not limited, the parameter will be the Empty Buffer.</td>
</tr>
<tr>
<td>TPM2B_NONCE</td>
<td>policyRef</td>
<td>a reference to a policy relating to the authorization — may be the Empty Buffer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Size is limited to be no larger than the nonce size supported on the TPM.</td>
</tr>
<tr>
<td>INT32</td>
<td>expiration</td>
<td>time when authorization will expire, measured in seconds from the time that nonceTPM was generated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If expiration is zero, a NULL Ticket is returned.</td>
</tr>
</tbody>
</table>

#### Table 116 — TPM2_PolicySecret Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_TIMEOUT</td>
<td>timeout</td>
<td>implementation-specific time value used to indicate to the TPM when the ticket expires; this ticket will use the TPMT_ST_AUTH_SECRET structure tag</td>
</tr>
<tr>
<td>TPMT_TK_AUTH</td>
<td>policyTicket</td>
<td>produced if the command succeeds and expiration in the command was non-zero</td>
</tr>
</tbody>
</table>
25.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicySecret_fp.h"
#include "Policy_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CPHASH</td>
<td>cpHash for policy was previously set to a value that is not the same as cpHashA</td>
</tr>
<tr>
<td>TPM_RC_EXPIRED</td>
<td>expiration indicates a time in the past</td>
</tr>
<tr>
<td>TPM_RC_NONCE</td>
<td>nonceTPM does not match the nonce associated with policySession</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>cpHashA is not the size of a digest for the hash associated with policySession</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>input policyID or expiration does not match the internal data in policy session</td>
</tr>
</tbody>
</table>

TPM_RC
TPM2_PolicySecret(
  PolicySecret_In *in,   // IN: input parameter list
  PolicySecret_Out *out  // OUT: output parameter list
) {
  TPM_RC result;
  SESSION *session;
  TPM2B_NAME entityName;
  UINT32 expiration = (in->expiration < 0) ? -(in->expiration) : in->expiration;
  UINT64 authTimeout = 0;

  // Input Validation
  if (session->attributes.isTrialPolicy == CLEAR) {
    if (expiration != 0)
      authTimeout = expiration * 1000 + session->startTime;
    result = PolicyParameterChecks(session, authTimeout,
      &in->cpHashA, &in->nonceTPM,
      RC_PolicySecret_nonceTPM,
      RC_PolicySecret_cpHashA,
      RC_PolicySecret_expiration);
    if (result != TPM_RC_SUCCESS)
      return result;
  }

  // Internal Data Update
  entityName.t.size = EntityGetName(in->authHandle, &entityName.t.name);
  PolicyContextUpdate(TPM_CC_PolicySecret, &entityName, &in->policyRef,
```
&in-&gt;cpHashA, authTimeout, session);

// Command Output

// Create ticket and timeout buffer if in-&gt;expiration &lt; 0 and this is not
// a trial session.
// NOTE: PolicyParameterChecks() makes sure that nonceTPM is present
// when expiration is non-zero.
if(   in-&gt;expiration &lt; 0
     &amp;&amp; session-&gt;attributes.isTrialPolicy == CLEAR
 )
{
  // Generate timeout buffer.  The format of output timeout buffer is
  // TPM-specific.
  // Note: can't do a direct copy because the output buffer is a byte
  // array and it may not be aligned to accept a 64-bit value.  The method
  // used has the side-effect of making the returned value a big-endian,
  // 64-bit value that is byte aligned.
  out-&gt;timeout.t.size = sizeof(UINT64);
  UINT64_TO_BYTE_ARRAY(authTimeout, out-&gt;timeout.t.buffer);

  // Compute policy ticket
  TicketComputeAuth(TPM_ST_AUTH_SECRET, EntityGetHierarchy(in-&gt;authHandle),
                     authTimeout, &in-&gt;cpHashA, &in-&gt;policyRef,
                     &entityName, &out-&gt;policyTicket);
}

else
{
  // timeout buffer is null
  out-&gt;timeout.t.size = 0;

  // auth ticket is null
  out-&gt;policyTicket.tag = TPM_ST_AUTH_SECRET;
  out-&gt;policyTicket.hierarchy = TPM_RH_NULL;
  out-&gt;policyTicket.digest.t.size = 0;
}

return TPM_RC_SUCCESS;
25.5  TPM2_PolicyTicket

25.5.1  General Description

This command is similar to TPM2_PolicySigned() except that it takes a ticket instead of a signed authorization. The ticket represents a validated authorization that had an expiration time associated with it.

The parameters of this command are checked as described in 25.2.2.

If the checks succeed, the TPM uses the timeout, cpHashA, policyRef, and keyName to construct a ticket to compare with the value in ticket. If these tickets match, then the TPM will create a TPM2B_NAME (objectName) using authName and update the context of policySession by PolicyUpdate() (see 25.2.3).

\[
\text{PolicyUpdate}(\text{commandCode}, \text{authName}, \text{policyRef})
\]

If the structure tag of ticket is TPM_ST_AUTH_SECRET, then \text{commandCode} will be TPM_CC_PolicySecret. If the structure tag of ticket is TPM_ST_AUTH_SIGNED, then \text{commandCode} will be TPM_CC_PolicySigned.

If the \text{cpHashA} command parameter is not an Empty Buffer, it may be copied to \text{cpHash} in the session context as described in 25.2.1.
25.5.2 Command and Response

Table 117 — TPM2_PolicyTicket Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyTicket</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_TIMEOUT</td>
<td>timeout</td>
<td>time when authorization will expire</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The contents are TPM specific. This shall be the value returned when ticket was produced.</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>cpHashA</td>
<td>digest of the command parameters to which this authorization is limited</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If it is not limited, the parameter will be the Empty Buffer.</td>
</tr>
<tr>
<td>TPM2B_NONCE</td>
<td>policyRef</td>
<td>reference to a qualifier for the policy – may be the Empty Buffer.</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>authName</td>
<td>name of the object that provided the authorization</td>
</tr>
<tr>
<td>TPMT_TK_AUTH</td>
<td>ticket</td>
<td>an authorization ticket returned by the TPM in response to a TPM2_PolicySigned() or TPM2_PolicySecret()</td>
</tr>
</tbody>
</table>

Table 118 — TPM2_PolicyTicket Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.5.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyTicket_fp.h"
#include "Policy_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CPHASH</td>
<td>policy's cpHash was previously set to a different value</td>
</tr>
<tr>
<td>TPM_RC_EXPIRED</td>
<td>timeout value in the ticket is in the past and the ticket has expired</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>timeout or cpHash has invalid size for the ticket</td>
</tr>
<tr>
<td>TPM_RC_TICKET</td>
<td>ticket is not valid</td>
</tr>
</tbody>
</table>

```TPM_RC

```c
TPM2_PolicyTicket(

```PolicyTicket_In *in  // IN: input parameter list

```c

<table>
<thead>
<tr>
<th>TPM_RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM2_PolicyTicket()</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>PolicyTicket_In *in</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>// IN: input parameter list</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>{</td>
</tr>
<tr>
<td>TPM_RC result;</td>
</tr>
<tr>
<td>SESSION *session;</td>
</tr>
<tr>
<td>UINT64 timeout;</td>
</tr>
<tr>
<td>TPMT_TK_AUTH ticketToCompare;</td>
</tr>
<tr>
<td>TPM_CC commandCode = TPM_CC_PolicySecret;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>// Input Validation</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>// Get pointer to the session structure</td>
</tr>
<tr>
<td>session = SessionGet(in-&gt;policySession);</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>// NOTE: A trial policy session is not allowed to use this command.</td>
</tr>
<tr>
<td>// A ticket is used in place of a previously given authorization. Since</td>
</tr>
<tr>
<td>// a trial policy doesn’t actually authenticate, the validated</td>
</tr>
<tr>
<td>// ticket is not necessary and, in place of using a ticket, one</td>
</tr>
<tr>
<td>// should use the intended authorization for which the ticket</td>
</tr>
<tr>
<td>// would be a substitute.</td>
</tr>
<tr>
<td>if(session-&gt;attributes.isTrialPolicy)</td>
</tr>
<tr>
<td>return TPM_RCS_ATTRIBUTES + RC_PolicyTicket_policySession;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>// Restore timeout data. The format of timeout buffer is TPM-specific.</td>
</tr>
<tr>
<td>// In this implementation, we simply copy the value of timeout to the</td>
</tr>
<tr>
<td>// buffer.</td>
</tr>
<tr>
<td>if(in-&gt;timeout.t.size != sizeof(UINT64))</td>
</tr>
<tr>
<td>return TPM_RC_SIZE + RC_PolicyTicket_timeout;</td>
</tr>
<tr>
<td>timeout = BYTE_ARRAY_TO_UINT64(in-&gt;timeout.t.buffer);</td>
</tr>
<tr>
<td>// Do the normal checks on the cpHashA and timeout values</td>
</tr>
<tr>
<td>result = PolicyParameterChecks(session, timeout,</td>
</tr>
<tr>
<td>&amp;in-&gt;cpHashA, NULL,</td>
</tr>
<tr>
<td>0, // no bad nonce return</td>
</tr>
<tr>
<td>RC_PolicyTicket_cpHashA,</td>
</tr>
<tr>
<td>RC_PolicyTicket_timeout);</td>
</tr>
<tr>
<td>if(result != TPM_RC_SUCCESS)</td>
</tr>
<tr>
<td>return result;</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>// Validate Ticket</td>
</tr>
<tr>
<td>// Re-generate policy ticket by input parameters</td>
</tr>
<tr>
<td>TicketComputeAuth(in-&gt;ticket.tag, in-&gt;ticket.hierarchy, timeout, &amp;in-&gt;cpHashA,</td>
</tr>
<tr>
<td>&amp;in-&gt;policyRef, &amp;in-&gt;authName, &amp;ticketToCompare);</td>
</tr>
</tbody>
</table>

```
50    // Compare generated digest with input ticket digest
51    if(!Memory2BEqual(&in->ticket.digest.b, &ticketToCompare.digest.b))
52        return TPM_RC_TICKET + RC_PolicyTicket_ticket;
53
54    // Internal Data Update
55
56    // Is this ticket to take the place of a TPM2_PolicySigned() or
57    // a TPM2_PolicySecret()?
58    if(in->ticket.tag == TPM_ST_AUTH_SIGNED)
59        commandCode = TPM_CC_PolicySigned;
60    else if(in->ticket.tag == TPM_ST_AUTH_SECRET)
61        commandCode = TPM_CC_PolicySecret;
62    else
63        // There could only be two possible tag values. Any other value should
64        // be caught by the ticket validation process.
65        pAssert(FALSE);
66
67    // Update policy context
68    PolicyContextUpdate(commandCode, &in->authName, &in->policyRef,
69                        &in->cpHashA, timeout, session);
70
71    return TPM_RC_SUCCESS;
72    }
25.6 TPM2_PolicyOR

25.6.1 General Description

This command allows options in authorizations without requiring that the TPM evaluate all of the options. If a policy may be satisfied by different sets of conditions, the TPM need only evaluate one set that satisfies the policy. This command will indicate that one of the required sets of conditions has been satisfied.

PolicySession→policyDigest is compared against the list of provided values. If the current PolicySession→policyDigest does not match any value in the list, the TPM shall return TPM_RC_VALUE. Otherwise, it will replace policySession→policyDigest with the digest of the concatenation of all of the digests and return TPM_RC_SUCCESS.

If policySession is a trial session, the TPM will assume that policySession→policyDigest matches one of the list entries and compute the new value of policyDigest.

The algorithm for computing the new value for policyDigest of policySession is:

a) Concatenate all the digest values in pHashList:

\[
digests := pHashList.digests[1].buffer || … || pHashList.digests[n].buffer
\]  

NOTE 1 The TPM makes no check to see if the size of an entry matches the size of the digest of the policy.

b) Reset policyDigest to a Zero Digest.

c) Extend the command code and the hashes computed in step a) above:

\[
policyDigest_{new} := H_{\text{PolicyAlg}}(policyDigest_{old} || TPM_{\text{CC PolicyOR}} || digests)
\]  

NOTE 2 The computation in b) and c) above is equivalent to:

\[
policyDigest_{new} := H_{\text{PolicyAlg}}(0…0 || TPM_{\text{CC PolicyOR}} || digests)
\]

A TPM shall support a list with at least eight tagged digest values.

NOTE 3 If policies are to be portable between TPMs, then they should not use more than eight values.
## 25.6.2 Command and Response

### Table 119 — TPM2_PolicyOR Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyOR, handle for the policy session being extended Auth Index: None</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td></td>
</tr>
<tr>
<td>TPML_DIGEST</td>
<td>pHashList</td>
<td>the list of hashes to check for a match</td>
</tr>
</tbody>
</table>

### Table 120 — TPM2_PolicyOR Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.6.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyOR_fp.h"
#include "Policy_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>no digest in pHashList matched the current</td>
</tr>
<tr>
<td></td>
<td>value of policyDigest for policySession</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_PolicyOR(
  PolicyOR_In *in         // IN: input parameter list
)
{
    SESSION *session;
    UINT32 i;

    // Input Validation and Update
    session = SessionGet(in->policySession);

    // Compare and Update Internal Session policy if match
    for (i = 0; i < in->pHashList.count; i++)
    {
        if (session->attributes.isTrialPolicy == SET ||
            Memory2BEqual(&session->u2.policyDigest.b,
                          &in->pHashList.digests[i].b))
        {
            // Found a match
            HASH_STATE hashState;
            TPM_CC commandCode = TPM_CC_PolicyOR;

            // Start hash
            session->u2.policyDigest.t.size = CryptStartHash(session->authHashAlg,
                                                            &hashState);

            // Set policyDigest to 0 string and add it to hash
            MemorySet(session->u2.policyDigest.t.buffer, 0,
                       session->u2.policyDigest.t.size);
            CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

            // Add command code
            CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

            // Add each of the hashes in the list
            for (i = 0; i < in->pHashList.count; i++)
            {
                // Extend policyDigest
                CryptUpdateDigest2B(&hashState, &in->pHashList.digests[i].b);
            }

            // Complete digest
            CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

            return TPM_RC_SUCCESS;
        }
    }

    // None of the values in the list matched the current policyDigest
    return TPM_RC_VALUE + RC_PolicyOR_pHashList;
```
54 }
25.7 TPM2_PolicyPCR

25.7.1 General Description

This command is used to cause conditional gating of a policy based on PCR. This allows one group of authorizations to occur when PCR are in one state and a different set of authorizations when the PCR are in a different state. If this command is used for a trial policySession, policySession→policyDigest will be updated using the values from the command rather than the values from digest of the TPM PCR.

The TPM will modify the pcrs parameter so that bits that correspond to unimplemented PCR are CLEAR. If policySession is not a trial policy session, the TPM will use the modified value of pcrs to select PCR values to hash according to Part 1, Selecting Multiple PCR. The hash algorithm of the policy session is used to compute a digest (digestTPM) of the selected PCR. If pcrDigest does not have a length of zero, then it is compared to digestTPM; and if the values do not match, the TPM shall return TPM_RC_VALUE and make no change to policySession→policyDigest. If the values match, or if the length of pcrDigest is zero, then policySession→policyDigest is extended by:

\[ policyDigest_{\text{new}} := H_{\text{policyAlg}}(policyDigest_{\text{old}} \ || \ TPM\_CC\_PolicyPCR \ || \ pcrs \ || \ digestTPM) \]  \hspace{1cm} (19)

where

- \( pcrs \) the pcrs parameter with bits corresponding to unimplemented PCR set to 0
- \( digestTPM \) the digest of the selected PCR using the hash algorithm of the policy session

NOTE 1 If the caller provides the expected PCR value, the intention is that the policy evaluation stop at that point if the PCR do not match. If the caller does not provide the expected PCR value, then the validity of the settings will not be determined until an attempt is made to use the policy for authorization. If the policy is constructed such that the PCR check comes before user authorization checks, this early termination would allow software to avoid unnecessary prompts for user input to satisfy a policy that would fail later due to incorrect PCR values.

After this command completes successfully, the TPM shall return TPM_RC_PCR_CHANGED if the policy session is used for authorization and the PCR are not known to be correct.

The TPM uses a “generation” number (pcrUpdateCounter) that is incremented each time PCR are updated (unless the PCR being changed is specified not to cause a change to this counter). The value of this counter is stored in the policy session context (policySession→pcrUpdateCounter) when this command is executed. When the policy is used for authorization, the current value of the counter is compared to the value in the policy session context and the authorization will fail if the values are not the same.

When this command is executed, policySession→pcrUpdateCounter is checked to see if it has been previously set (in the reference implementation, it has a value of zero if not previously set). If it has been set, it will be compared with the current value of pcrUpdateCounter to determine if any PCR changes have occurred. If the values are different, the TPM shall return TPM_RC_PCR_CHANGED. If policySession→pcrUpdateCounter has not been set, then it is set to the current value of pcrUpdateCounter.

If policySession is a trial policy session, the TPM will not check any PCR and will compute:

\[ policyDigest_{\text{new}} := H_{\text{policyAlg}}(policyDigest_{\text{old}} \ || \ TPM\_CC\_PolicyPCR \ || \ pcrs \ || \ pcrDigest) \]  \hspace{1cm} (20)

In this computation, pcrs is the input parameter without modification.

NOTE 2 The pcrs parameter is expected to match the configuration of the TPM for which the policy is being computed which may not be the same as the TPM on which the trial policy is being computed.
25.7.2 Command and Response

Table 121 — TPM2_PolicyPCR Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyPCR</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td>Auth Index: None</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>pcrDigest</td>
<td>expected digest value of the selected PCR using the hash algorithm of the session; may be zero length</td>
</tr>
<tr>
<td>TPML_PCR_SELECTION</td>
<td>pcrs</td>
<td>the PCR to include in the check digest</td>
</tr>
</tbody>
</table>

Table 122 — TPM2_PolicyPCR Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.7.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyPCR_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>if provided, pcrDigest does not match the current PCR settings</td>
</tr>
<tr>
<td>TPM_RC_PCR_CHANGED</td>
<td>a previous TPM2_PolicyPCR() set pcrCounter and it has changed</td>
</tr>
</tbody>
</table>

TPM_RC
TPM2_PolicyPCR(
    PolicyPCR_In *in  // IN: input parameter list
) {
    SESSION *session;
    TPM2B_DIGEST pcrDigest;
    BYTE     pcrs[sizeof(TPML_PCR_SELECTION)];
    UINT32   pcrSize;
    BYTE     *buffer;
    TPM_CC   commandCode = TPM_CC_PolicyPCR;
    HASH_STATE hashState;

    // Input Validation
    session = SessionGet(in->policySession);

    // Do validation for non trial session
    if(session->attributes.isTrialPolicy == CLEAR)
        {
            // Make sure that this is not going to invalidate a previous PCR check
            if(session->pcrCounter != 0 && session->pcrCounter != gr.pcrCounter)
                return TPM_RC_PCR_CHANGED;

            // Compute current PCR digest
            PCRComputeCurrentDigest(session->authHashAlg, &in->pcrs, &pcrDigest);

            // If the caller specified the PCR digest and it does not
            // match the current PCR settings, return an error..
            if(in->pcrDigest.t.size != 0)
                {
                    if(!Memory2BEqual(&in->pcrDigest.b, &pcrDigest.b))
                        return TPM_RC_VALUE + RC_PolicyPCR_pcrDigest;
                }
        }
    else
        {
            // For trial session, just use the input PCR digest
            pcrDigest = in->pcrDigest;
        }

    // Internal Data Update
    // Update policy hash
    // policyDigestnew = hash(  policyDigestold || TPM_CC_PolicyPCR
    //                         || pcrs || pcrDigest)
    // Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    // add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
```
// add commandCode
CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

// add PCRs
buffer = pcrs;
pcrSize = TPML_PCR_SELECTION_Marshal(&in->pcrs, &buffer, NULL);
CryptUpdateDigest(&hashState, pcrSize, pcrs);

// add PCR digest
CryptUpdateDigest2B(&hashState, &pcrDigest.b);

// complete the hash and get the results
CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

// update pcrCounter in session context for non trial session
if(session->attributes.isTrialPolicy == CLEAR)
{
    session->pcrCounter = gr.pcrCounter;
}

return TPM_RC_SUCCESS;
25.8 TPM2_PolicyLocality

25.8.1 General Description

This command indicates that the authorization will be limited to a specific locality.

\( \text{policySession} \rightarrow \text{commandLocality} \) is a parameter kept in the session context. It is initialized when the policy session is started to allow the policy to apply to any locality.

If \( \text{locality} \) has a value greater than 31, then an extended locality is indicated. For an extended locality, the TPM will validate that \( \text{policySession} \rightarrow \text{commandLocality} \) is has not previously been set or that the current value of \( \text{policySession} \rightarrow \text{commandLocality} \) is the same as \( \text{locality} \) (TPM_RC_RANGE).

When \( \text{locality} \) is not an extended locality, the TPM will validate that the \( \text{policySession} \rightarrow \text{commandLocality} \) is not set or is not set to an extended locality value (TPM_RC_RANGE). If not the TPM will disable any locality not SET in the \( \text{locality} \) parameter. If the result of disabling localities results in no locality being enabled, the TPM will return TPM_RC_RANGE.

If no error occurred in the validation of \( \text{locality} \), \( \text{policySession} \rightarrow \text{policyDigest} \) is extended with

\[
\text{policyDigest}_{\text{new}} := H_{\text{policyAlg}}(\text{policyDigest}_{\text{old}} | | \text{TPM_CC_PolicyLocality} | | \text{locality})
\]  

(21)

Then \( \text{policySession} \rightarrow \text{commandLocality} \) is updated to indicate which localities are still allowed after execution of TPM2_PolicyLocality().

When the policy session is used to authorize a command, the authorization will fail if the locality used for the command is not one of the enabled localities in \( \text{policySession} \rightarrow \text{commandLocality} \).
25.8.2 Command and Response

Table 123 — TPM2_PolicyLocality Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyLocality</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPMA_LOCALITY</td>
<td>locality</td>
<td>the allowed localities for the policy</td>
</tr>
</tbody>
</table>

Table 124 — TPM2_PolicyLocality Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.8.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyLocality_fp.h"

Limit a policy to a specific locality

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_RANGE</td>
<td>all the locality values selected by <code>locality</code> have been disabled by previous TPM2_PolicyLocality() calls.</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_PolicyLocality(
    PolicyLocality_In   *in   // IN: input parameter list
)
{
    SESSION     *session;
    BYTE         marshalBuffer[sizeof(TPMA_LOCALITY)];
    BYTE         prevSetting[sizeof(TPMA_LOCALITY)];
    UINT32       marshalSize;
    BYTE        *buffer;
    TPM_CC       commandCode = TPM_CC_PolicyLocality;
    HASH_STATE   hashState;

    // Input Validation
    // Get pointer to the session structure
    session = SessionGet(in->policySession);
    // Get new locality setting in canonical form
    buffer = marshalBuffer;
    marshalSize = TPMA_LOCALITY_Marshal(&in->locality, &buffer, NULL);
    // Its an error if the locality parameter is zero
    if(marshalBuffer[0] == 0)
        return TPM_RC_RANGE + RC_PolicyLocality_locality;
    // Get existing locality setting in canonical form
    buffer = prevSetting;
    TPMA_LOCALITY_Marshal(&session->commandLocality, &buffer, NULL);
    // If the locality has been previously set, then it needs to be the same
    // tye as the input locality (i.e. both extended or both normal
    if(prevSetting[0] != 0 && ((prevSetting[0] <= 0) != (marshalBuffer[0] <= 0)))
        return TPM_RC_RANGE + RC_PolicyLocality_locality;
    // See if the input is a regular or extended locality
    if(marshalBuffer[0] < 32)
    {
        // For regular locality
        // The previous setting must not be an extended locality
        if(prevSetting[0] > 31)
            return TPM_RC_RANGE + RC_PolicyLocality_locality;
        // if there was no previous setting, start with all normal localities
        // enabled
        if(prevSetting[0] == 0)
            prevSetting[0] = 0x1F;
        // AND the new setting with the previous setting and store it in prevSetting
```
prevSetting[0] &= marshalBuffer[0];

// The result setting can not be 0
if (prevSetting[0] == 0)
    return TPM_RC_RANGE + RC_PolicyLocality_locality;
else
{
    // for extended locality
    // if the locality has already been set, then it must match the
    if (prevSetting[0] != 0 && prevSetting[0] != marshalBuffer[0])
        return TPM_RC_RANGE + RC_PolicyLocality_locality;

    // Setting is OK
    prevSetting[0] = marshalBuffer[0];
}

// Internal Data Update

// Update policy hash
// policyDigestnew = hash(policyDigestold || TPM_CC_PolicyLocality || locality)
// Start hash
CryptStartHash(session->authHashAlg, &hashState);

// add old digest
CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

// add commandCode
CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

// add input locality
CryptUpdateDigest(&hashState, marshalSize, marshalBuffer);

// complete the digest
CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

// update session locality by unmarshal function. The function must succeed
// because both input and existing locality setting have been validated.
buffer = prevSetting;
TPMA_LOCALITY_Unmarshal(&session->commandLocality, &buffer,
    (INT32 *) &marshalSize);
return TPM_RC_SUCCESS;
25.9 TPM2_PolicyNV

25.9.1 General Description

This command is used to cause conditional gating of a policy based on the contents of an NV Index.

If policySession is a trial policy session, the TPM will update policySession→policyDigest as shown in equations (22) and (23) below and return TPM_RC_SUCCESS. It will not perform any validation. The remainder of this general description would apply only if policySession is not a trial policy session.

An authorization session providing authorization to read the NV Index shall be provided.

NOTE 1  If read access is controlled by policy, the policy should include a branch that authorizes a TPM2_PolicyNV().

If TPMA_NV_WRITTEN is not SET in the NV Index, the TPM shall return TPM_RC_NV_UNINITIALIZED.

The TPM will validate that the size of operandB plus offset is not greater than the size of the NV Index. If it is, the TPM shall return TPM_RC_SIZE.

The TPM will perform the indicated arithmetic check on the indicated portion of the selected NV Index. If the check fails, the TPM shall return TPM_RC_POLICY and not change policySession→policyDigest. If the check succeeds, the TPM will hash the arguments:

\[
\text{args} := \text{H}_{\text{policyAlg}}(\text{operand.buffer} || \text{offset} || \text{operation})
\]  

where

- \( \text{H}_{\text{policyAlg}}() \): hash function using the algorithm of the policy session
- \( \text{operandB} \): the value used for the comparison
- \( \text{offset} \): offset from the start of the NV Index data to start the comparison
- \( \text{operation} \): the operation parameter indicating the comparison being performed

The value of args and the Name of the NV Index are extended to policySession→policyDigest by

\[
\text{policyDigest}_{\text{new}} := \text{H}_{\text{policyAlg}}(\text{policyDigest}_{\text{old}} || \text{TPM_CC_PolicyNV} || \text{args} || \text{nvIndex}→\text{Name})
\]

where

- \( \text{H}_{\text{policyAlg}}() \): hash function using the algorithm of the policy session
- \( \text{args} \): value computed in equation (22)
- \( \text{nvIndex}→\text{Name} \): the Name of the NV Index

The signed arithmetic operations are performed using twos-compliment.

Magnitude comparisons assume that the octet at offset zero in the referenced NV location and in operandB contain the most significant octet of the data.

NOTE 2  When an Index is written, it has a different authorization name than an Index that has not been written. It is possible to use this change in the NV Index to create a write-once Index.
25.9.2 Command and Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyNV handle indicating the source of the authorization value</td>
</tr>
<tr>
<td>TPMI_RH_NV_AUTH</td>
<td>@authHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>the NV Index of the area to read</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_OPERAND</td>
<td>operandB</td>
<td>the second operand</td>
</tr>
<tr>
<td>UINT16</td>
<td>offset</td>
<td>the offset in the NV Index for the start of operand A</td>
</tr>
<tr>
<td>TPM_EO</td>
<td>operation</td>
<td>the comparison to make</td>
</tr>
</tbody>
</table>

Table 126 — TPM2_PolicyNV Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
### 25.9.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyNV_fp.h"
#include "Policy_spt_fp.h"
#include "NV_spt_fp.h" // Include NV support routine for read access check
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_AUTH_TYPE</td>
<td>NV index authorization type is not correct</td>
</tr>
<tr>
<td>TPM_RC_NV_LOCKED</td>
<td>NV index read locked</td>
</tr>
<tr>
<td>TPM_RC_NV_UNINITIALIZED</td>
<td>the NV index has not been initialized</td>
</tr>
<tr>
<td>TPM_RC_POLICY</td>
<td>the comparison to the NV contents failed</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>the size of <code>nvIndex</code> data starting at <code>offset</code> is less than the size of operandB</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_PolicyNV(
    PolicyNV_In         *in  // IN: input parameter list
) {
    TPM_RC               result;
    SESSION             *session;
    NV_INDEX             nvIndex;
    BYTE             nvBuffer[sizeof(in->operandB.t.buffer)];
    TPM2B_NAME           nvName;
    TPM_CC               commandCode = TPM_CC_PolicyNV;
    HASH_STATE           hashState;
    TPM2B_DIGEST         argHash;

    // Input Validation

    // Get NV index information
    NvGetIndexInfo(in->nvIndex, &nvIndex);
    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // If this is a trial policy, skip all validations and the operation
    if(session->attributes.isTrialPolicy == CLEAR) {
        // NV Read access check. NV index should be allowed for read. A
        // TPM_RC_AUTH_TYPE or TPM_RC_NV_LOCKED error may be return at this
        // point
        result = NvReadAccessChecks(in->authHandle, in->nvIndex);
        if(result != TPM_RC_SUCCESS) return result;

        // Valid NV data size should not be smaller than input operandB size
        if((nvIndex.publicArea.dataSize - in->offset) < in->operandB.t.size) return TPM_RC_SIZE + RC_PolicyNV_operandB;

        // Arithmetic Comparison

        // Get NV data. The size of NV data equals the input operand B size
        NvGetIndexData(in->nvIndex, &nvIndex, in->offset,
                       in->operandB.t.size, nvBuffer);

        switch(in->operation)
```
{  
    case TPM_EO_EQ:  
        // compare A = B  
        if (CryptCompare(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) != 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_NEQ:  
        // compare A ! = B  
        if (CryptCompare(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) == 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_SIGNED_GT:  
        // compare A > B signed  
        if (CryptCompareSigned(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) <= 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_UNSIGNED_GT:  
        // compare A > B unsigned  
        if (CryptCompare(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) <= 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_SIGNED_LT:  
        // compare A < B signed  
        if (CryptCompareSigned(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) >= 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_UNSIGNED_LT:  
        // compare A < B unsigned  
        if (CryptCompare(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) >= 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_SIGNED_GE:  
        // compare A >= B signed  
        if (CryptCompareSigned(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) < 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_UNSIGNED_GE:  
        // compare A >= B unsigned  
        if (CryptCompare(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) < 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_SIGNED_LE:  
        // compare A <= B signed  
        if (CryptCompareSigned(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) > 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_UNSIGNED_LE:  
        // compare A <= B unsigned  
        if (CryptCompare(in->operandB.t.size, nvBuffer,  
            in->operandB.t.size, in->operandB.t.buffer) > 0)  
            return TPM_RC_POLICY;  
        break;  
    case TPM_EO_BITSET:  
        // All bits SET in B are SET in A. ((A&B)=B)  
        {  
            // Copy B to buffer  
            CryptCopy(in->operandB.t.size, nvBuffer,  
                in->operandB.t.size, in->operandB.t.buffer)  

        return TPM_RC_POLICY;  
        }  
    break;  
}
# Part 3: Commands

## Trusted Platform Module Library

```c
UINT32 i;
for (i = 0; i < in->operandB.t.size; i++)
    if ((nvBuffer[i] & in->operandB.t.buffer[i])
        != in->operandB.t.buffer[i])
        return TPM_RC_POLICY;
break;
case TPM_EO_BITCLEAR:
    // All bits SET in B are CLEAR in A. ((A&B)=0)
    {
        UINT32 i;
        for (i = 0; i < in->operandB.t.size; i++)
            if ((nvBuffer[i] & in->operandB.t.buffer[i]) != 0)
                return TPM_RC_POLICY;
        break;
    default:
        pAssert(FALSE);
        break;
    }
// Internal Data Update
// Start argument hash
argHash.t.size = CryptStartHash(session->authHashAlg, &hashState);
// add operandB
CryptUpdateDigest2B(&hashState, &in->operandB.b);
// add offset
CryptUpdateDigestInt(&hashState, sizeof(UINT16), &in->offset);
// add operation
CryptUpdateDigestInt(&hashState, sizeof(TPM_EO), &in->operation);
// complete argument digest
CryptCompleteHash2B(&hashState, &argHash.b);
// Update policyDigest
// Start digest
CryptStartHash(session->authHashAlg, &hashState);
// add old digest
CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
// add commandCode
CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
// add argument digest
CryptUpdateDigest2B(&hashState, &argHash.b);
// Adding nvName
nvName.t.size = EntityGetName(in->nvIndex, &nvName.t.name);
CryptUpdateDigest2B(&hashState, &nvName.b);
// complete the digest
CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
return TPM_RC_SUCCESS;
```
25.10 TPM2_PolicyCounterTimer

25.10.1 General Description

This command is used to cause conditional gating of a policy based on the contents of the TPMS_TIME_INFO structure.

If policySession is a trial policy session, the TPM will update policySession→policyDigest as shown in equations (24) and (25) below and return TPM_RC_SUCCESS. It will not perform any validation. The remainder of this general description would apply only if policySession is not a trial policy session.

The TPM will perform the indicated arithmetic check on the indicated portion of the TPMS_TIME_INFO structure. If the check fails, the TPM shall return TPM_RC_POLICY and not change policySession→policyDigest. If the check succeeds, the TPM will hash the arguments:

\[ args \triangleq H_{policyAlg}(operandB.buffer |\| offset |\| operation) \]  (24)

where

- \( H_{policyAlg()} \) hash function using the algorithm of the policy session
- \( operandB.buffer \) the value used for the comparison
- \( offset \) offset from the start of the TPMS_TIME_INFO structure at which the comparison starts
- \( operation \) the operation parameter indicating the comparison being performed

The value of \( args \) is extended to policySession→policyDigest by

\[ policyDigest_{new} \triangleq H_{policyAlg}(policyDigest_{old} |\| TPM_CC_PolicyCounterTimer |\| args) \]  (25)

where

- \( H_{policyAlg()} \) hash function using the algorithm of the policy session
- \( args \) value computed in equation (24)

The signed arithmetic operations are performed using twos-compliment.

Magnitude comparisons assume that the octet at offset zero in the referenced location and in operandB contain the most significant octet of the data.
### 25.10.2 Command and Response

**Table 127 — TPM2_PolicyCounterTimer Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyCounterTimer handle for the policy session being extended</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_OPERAND</td>
<td>operandB</td>
<td>the second operand</td>
</tr>
<tr>
<td>UINT16</td>
<td>offset</td>
<td>the offset in TPMS_TIME_INFO structure for the start of operand A</td>
</tr>
<tr>
<td>TPM_EO</td>
<td>operation</td>
<td>the comparison to make</td>
</tr>
</tbody>
</table>

**Table 128 — TPM2_PolicyCounterTimer Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.10.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyCounterTimer_fp.h"
#include "Policy_spt_fp.h"
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_POLICY</td>
<td>the comparison of the selected portion of the TPMS_TIME_INFO with operandB failed</td>
</tr>
<tr>
<td>TPM_RC_RANGE</td>
<td>offset + size exceed size of TPMS_TIME_INFO structure</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_PolicyCounterTimer(
PolicyCounterTimer_In   *in   // IN: input parameter list
)
{
  TPM_RC               result;
  SESSION             *session;
  TIME_INFO            infoData;   // data buffer of TPMS_TIME_INFO
  TPM_CC               commandCode = TPM_CC_PolicyCounterTimer;
  HASH_STATE           hashState;
  TPM2B_DIGEST         argHash;

  // Input Validation
  if(in-offset < sizeof(UINT64) + sizeof(UINT64))
  {
    // Using Clock or Time so see if clock is running. Clock doesn't run while NV is unavailable.
    // TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned here.
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
      return result;
  }

  // Get pointer to the session structure
  session = SessionGet(in-policySession);

  // If this is a trial policy, skip all validations and the operation
  if(session->attributes.isTrialPolicy == CLEAR)
  {
    // Get time data info. The size of time info data equals the input
    // operand B size. A TPM_RC_RANGE error may be returned at this point
    result = TimeGetRange(in-offset, in->operandB.t.size, &infoData);
    if(result != TPM_RC_SUCCESS) return result;

    // Arithmetic Comparison
    switch(in->operation)
    {
      case TPM_EQ_EQ:
        // compare A = B
        if(CryptCompare(in->operandB.t.size, infoData,
          in->operandB.t.size, in->operandB.t.buffer) != 0)
          return TPM_RC_POLICY;
        break;
      case TPM_EQ_NEQ:
        // compare A != B
        if(CryptCompare(in->operandB.t.size, infoData,
          in->operandB.t.size, in->operandB.t.buffer) != 0)
          return TPM_RC_POLICY;
        break;
    }
  }
```

53          return TPM_RC_POLICY;
54      break;
55    case TPM_EO_SIGNED_GT:
56      // compare A > B signed
57      if(CryptCompareSigned(in->operandB.t.size, infoData,
58        in->operandB.t.size, in->operandB.t.buffer) <= 0)
59          return TPM_RC_POLICY;
60      break;
61    case TPM_EO_UNSIGNED_GT:
62      // compare A > B unsigned
63      if(CryptCompare(in->operandB.t.size, infoData,
64        in->operandB.t.size, in->operandB.t.buffer) <= 0)
65          return TPM_RC_POLICY;
66      break;
67    case TPM_EO_SIGNED_LT:
68      // compare A < B signed
69      if(CryptCompareSigned(in->operandB.t.size, infoData,
70        in->operandB.t.size, in->operandB.t.buffer) >= 0)
71          return TPM_RC_POLICY;
72      break;
73    case TPM_EO_UNSIGNED_LT:
74      // compare A < B unsigned
75      if(CryptCompare(in->operandB.t.size, infoData,
76        in->operandB.t.size, in->operandB.t.buffer) >= 0)
77          return TPM_RC_POLICY;
78      break;
79    case TPM_EO_SIGNED_GE:
80      // compare A >= B signed
81      if(CryptCompareSigned(in->operandB.t.size, infoData,
82        in->operandB.t.size, in->operandB.t.buffer) < 0)
83          return TPM_RC_POLICY;
84      break;
85    case TPM_EO_UNSIGNED_GE:
86      // compare A >= B unsigned
87      if(CryptCompare(in->operandB.t.size, infoData,
88        in->operandB.t.size, in->operandB.t.buffer) < 0)
89          return TPM_RC_POLICY;
90      break;
91    case TPM_EO_SIGNED_LE:
92      // compare A <= B signed
93      if(CryptCompareSigned(in->operandB.t.size, infoData,
94        in->operandB.t.size, in->operandB.t.buffer) > 0)
95          return TPM_RC_POLICY;
96      break;
97    case TPM_EO_UNSIGNED_LE:
98      // compare A <= B unsigned
99      if(CryptCompare(in->operandB.t.size, infoData,
100        in->operandB.t.size, in->operandB.t.buffer) > 0)
101          return TPM_RC_POLICY;
102      break;
103    case TPM_EO_BITSET:
104      // All bits SET in B are SET in A. ((A&B)=B)
105      {
106        UINT32 i;
107        for (i = 0; i < in->operandB.t.size; i++)
108            if (infoData[i] & in->operandB.t.buffer[i])
109                return TPM_RC_POLICY;
110      }
111      break;
112    case TPM_EO_BITCLEAR:
113      // All bits SET in B are CLEAR in A. ((A&B)=0)
114      {
115          TPM_RC_POLICY;
116      }
UINT32 i;
for (i = 0; i < in->operandB.t.size; i++)
    if ((infoData[i] & in->operandB.t.buffer[i]) != 0)
        return TPM_RC_POLICY;
break;
default:
    pAssert(FALSE);
    break;
}
}

// Internal Data Update

// Start argument list hash
argHash.t.size = CryptStartHash(session->authHashAlg, &hashState);
// add operandB
CryptUpdateDigest2B(&hashState, &in->operandB.b);
// add offset
CryptUpdateDigestInt(&hashState, sizeof(UINT16), &in->offset);
// add operation
CryptUpdateDigestInt(&hashState, sizeof(TPM_EO), &in->operation);
// complete argument hash
CryptCompleteHash2B(&hashState, &argHash.b);

// update policyDigest
// start hash
CryptStartHash(session->authHashAlg, &hashState);
// add old digest
CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
// add commandCode
CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
// add argument digest
CryptUpdateDigest2B(&hashState, &argHash.b);
// complete the digest
CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
return TPM_RC_SUCCESS;
25.11 TPM2_PolicyCommandCode

25.11.1 General Description

This command indicates that the authorization will be limited to a specific command code.

If \( \text{policySession} \rightarrow \text{commandCode} \) has its default value, then it will be set to \( \text{code} \). If \( \text{policySession} \rightarrow \text{commandCode} \) does not have its default value, then the TPM will return TPM_RC_VALUE if the two values are not the same.

If \( \text{code} \) is not implemented, the TPM will return TPM_RC_POLICY_CC.

If the TPM does not return an error, it will update \( \text{policySession} \rightarrow \text{policyDigest} \) by

\[
\text{policyDigest}_{\text{new}} := \text{H}_{\text{policyAlg}}(\text{policyDigest}_{\text{old}} || \text{TPM_CC_PolicyCommandCode} || \text{code})
\]  

(26)

NOTE 1 If a previous TPM2_PolicyCommandCode() had been executed, then it is probable that the policy expression is improperly formed but the TPM does not return an error.

NOTE 2 A TPM2_PolicyOR() would be used to allow an authorization to be used for multiple commands.

When the policy session is used to authorize a command, the TPM will fail the command if the \( \text{commandCode} \) of that command does not match \( \text{policySession} \rightarrow \text{commandCode} \).

This command, or TPM2_PolicyDuplicationSelect(), is required to enable the policy to be used for ADMIN role authorization.

EXAMPLE Before TPM2_Certify() can be executed, TPM2_PolicyCommandCode() with code set to TPM_CC_Certify is required.
### 25.11.2 Command and Response

#### Table 129 — TPM2_PolicyCommandCode Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyCommandCode</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM_CC</td>
<td>code</td>
<td>the allowed commandCode</td>
</tr>
</tbody>
</table>

#### Table 130 — TPM2_PolicyCommandCode Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.11.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyCommandCode_fp.h"

Error Returns | Meaning
---|---
TPM_RC_VALUE | commandCode of policySession previously set to a different value

TPM2_PolicyCommandCode(
    PolicyCommandCode_In *in  // IN: input parameter list
) {
    SESSION  *session;
    TPM_CC    commandCode = TPM_CC_PolicyCommandCode;
    HASH_STATE hashState;

    // Input validation
    if(session->commandCode != 0 && session->commandCode != in->code)
        return TPM_RC_VALUE + RC_PolicyCommandCode_code;
    if(!CommandIsImplemented(in->code))
        return TPM_RC_POLICY_CC + RC_PolicyCommandCode_code;

    // Internal Data Update
    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCommandCode || code)
    // Start hash
    CryptStartHash(session->authHashAlg, &hashState);
    // add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
    // add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
    // add input commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &in->code);
    // complete the hash and get the results
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
    // update commandCode value in session context
    session->commandCode = in->code;
    return TPM_RC_SUCCESS;
}
```
25.12 TPM2_PolicyPhysicalPresence

25.12.1 General Description

This command indicates that physical presence will need to be asserted at the time the authorization is performed.

If this command is successful, $policySession\rightarrow isPPRequired$ will be SET to indicate that this check is required when the policy is used for authorization. Additionally, $policySession\rightarrow policyDigest$ is extended with

\[
policyDigest_{new} := H_{policyAlg}(policyDigest_{old} || TPM_CC_PolicyPhysicalPresence)
\]
### 25.12.2 Command and Response

**Table 131 — TPM2_PolicyPhysicalPresence Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyPhysicalPresence</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended Auth Index: None</td>
</tr>
</tbody>
</table>

**Table 132 — TPM2_PolicyPhysicalPresence Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.12.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyPhysicalPresence_fp.h"
TPM_RC
TPM2_PolicyPhysicalPresence(
    PolicyPhysicalPresence_In *in // IN: input parameter list
) {
    SESSION *session;
    TPM_CC commandCode = TPM_CC_PolicyPhysicalPresence;
    HASH_STATE hashState;

    // Internal Data Update
    
    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyPhysicalPresence)
    // Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    // add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    // add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    // complete the digest
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    // update session attribute
    session->attributes.isPPRequired = SET;

    return TPM_RC_SUCCESS;
}
```
25.13 TPM2_PolicyCpHash

25.13.1 General Description

This command is used to allow a policy to be bound to a specific command and command parameters. TPM2_PolicySigned(), TPM2_PolicySecret(), and TPM2_PolicyTicket() are designed to allow an authorizing entity to execute an arbitrary command as the cpHashA parameter of those commands is not included in policySession→policyDigest. TPM2_PolicyCommandCode() allows the policy to be bound to a specific Command Code so that only certain entities may authorize specific command codes. This command allows the policy to be restricted such that an entity may only authorize a command with a specific set of parameters.

If policySession→cpHash is already set and not the same as cpHashA, then the TPM shall return TPM_RC_VALUE. If cpHashA does not have the size of the policySession→policyDigest, the TPM shall return TPM_RC_SIZE.

If the cpHashA checks succeed, policySession→cpHash is set to cpHashA and policySession→policyDigest is updated with

\[ policyDigest_{\text{new}} := H_{policyAlg}(policyDigest_{\text{old}} || TPM\_CC\_PolicyCpHash || cpHashA) \]
25.13.2 Command and Response

Table 133 — TPM2_PolicyCpHash Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyCpHash handle for the policy session being extended</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>cpHashA</td>
<td>the cpHash added to the policy</td>
</tr>
</tbody>
</table>

Table 134 — TPM2_PolicyCpHash Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.13.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyCpHash_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CPHASH</td>
<td>\textit{cpHash} of \textit{policySession} has previously been set to a different value</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>\textit{cpHashA} is not the size of a digest produced by the hash algorithm associated with \textit{policySession}</td>
</tr>
</tbody>
</table>

```c
TPM_RC

TPM2_PolicyCpHash(
    PolicyCpHash_In *in           // IN: input parameter list
)
{
    SESSION *session;
    TPM_CC commandCode = TPM_CC_PolicyCpHash;
    HASH_STATE hashState;

    // Input Validation
    // Get pointer to the session structure
    session = SessionGet(in->policySession);
    // A new cpHash is given in input parameter, but cpHash in session context
    // is not empty, or is not the same as the new cpHash
    if (    in->cpHashA.t.size != 0
        && session->u1.cpHash.t.size != 0
        && !Memory2BEqual(&in->cpHashA.b, &session->u1.cpHash.b)
    )
        return TPM_RC_CPHASH;
    // A valid cpHash must have the same size as session hash digest
    if (in->cpHashA.t.size != CryptGetHashDigestSize(session->authHashAlg))
        return TPM_RC_SIZE + RC_PolicyCpHash_cpHashA;

    // Internal Data Update
    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyCpHash || cpHashA)
    // Start hash
    CryptStartHash(session->authHashAlg, &hashState);
    // add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
    // add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
    // add cpHashA
    CryptUpdateDigest2B(&hashState, &in->cpHashA.b);
    // complete the digest and get the results
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
    // update cpHash in session context
    session->u1.cpHash = in->cpHashA;
    session->attributes.iscpHashDefined = SET;
    return TPM_RC_SUCCESS;
```
25.14 **TPM2_PolicyNameHash**

25.14.1 **General Description**

This command allows a policy to be bound to a specific set of TPM entities without being bound to the parameters of the command. This is most useful for commands such as TPM2_Duplicate() and for TPM2_PCR_Event() when the referenced PCR requires a policy.

The *nameHash* parameter should contain the digest of the Names associated with the handles to be used in the authorized command.

**EXAMPLE** For the TPM2_Duplicate() command, two handles are provided. One is the handle of the object being duplicated and the other is the handle of the new parent. For that command, *nameHash* would contain:

\[ nameHash := H_{policyAlg} (objectHandle \rightarrow Name || newParentHandle \rightarrow Name) \]

If *policySession*→*cpHash* is already set, the TPM shall return TPM_RC_VALUE. If the size of *nameHash* is not the size of *policySession*→*policyDigest*, the TPM shall return TPM_RC_SIZE. Otherwise, *policySession*→*cpHash* is set to *nameHash*.

If this command completes successfully, the *cpHash* of the authorized command will not be used for validation. Only the digest of the Names associated with the handles in the command will be used.

**NOTE 1** This allows the space normally used to hold *policySession*→*cpHash* to be used for *policySession*→*nameHash* instead.

The *policySession*→*policyDigest* will be updated with

\[ policyDigest_{new} := H_{policyAlg} (policyDigest_{old} | TPM_CC_PolicyNameHash | nameHash) \]

**NOTE 2** This command will often be used with TPM2_PolicyAuthorize() where the owner of the object being duplicated provides approval for their object to be migrated to a specific new parent.
25.14.2 Command and Response

Table 135 — TPM2_PolicyNameHash Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyNameHash handle for the policy session being extended</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>nameHash</td>
<td>the digest to be added to the policy</td>
</tr>
</tbody>
</table>

Table 136 — TPM2_PolicyNameHash Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
# 25.14.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyNameHash_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CPHASH</td>
<td><code>nameHash</code> has been previously set to a different value</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td><code>nameHash</code> is not the size of the digest produced by the hash algorithm associated with <code>policySession</code></td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_PolicyNameHash(
  PolicyNameHash_In  *in  // IN: input parameter list
)
{
  SESSION             *session;
  TPM_CC               commandCode = TPM_CC_PolicyNameHash;
  HASH_STATE           hashState;

  // Input Validation
  
  // Get pointer to the session structure
  session = SessionGet(in->policySession);

  // A new `nameHash` is given in input parameter, but `cpHash` in session context
  // is not empty
  if(in->nameHash.t.size != 0 && session->u1.cpHash.t.size != 0)
    return TPM_RC_CPHASH;

  // A valid `nameHash` must have the same size as session hash digest
  if(in->nameHash.t.size != CryptGetHashDigestSize(session->authHashAlg))
    return TPM_RC_SIZE + RC_PolicyNameHash_nameHash;

  // Internal Data Update

  // Update policy hash
  // `policyDigestnew` = hash(`policyDigestold` || TPM_CC_PolicyNameHash || `nameHash`)
  // Start hash
  CryptStartHash(session->authHashAlg, &hashState);

  // add old digest
  CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

  // add `commandCode`
  CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

  // add `nameHash`
  CryptUpdateDigest2B(&hashState, &in->nameHash.b);

  // complete the digest
  CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

  // clear iscpHashDefined bit to indicate now this field contains a `nameHash`
  session->attributes.iscpHashDefined = CLEAR;

  // update `nameHash` in session context
  session->u1.cpHash = in->nameHash;

  return TPM_RC_SUCCESS;
}
```
25.15 TPM2_PolicyDuplicationSelect

25.15.1 General Description

This command allows qualification of duplication to allow duplication to a selected new parent.

If this command not used in conjunction with TPM2_PolicyAuthorize(), then only the new parent is selected.

EXAMPLE
When an object is created when the list of allowed duplication targets is known, the policy would be created with includeObject CLEAR.

NOTE 1
Only the new parent may be selected because, without TPM2_PolicyAuthorize(), the Name of the Object to be duplicated would need to be known at the time that Object’s policy is created. However, since the Name of the Object includes its policy, the Name is not known.

If used in conjunction with TPM2_PolicyAuthorize(), then the authorizer of the new policy has the option of selecting just the new parent or of selecting both the new parent and the duplication Object.

NOTE 2
If the authorizing entity for an TPM2_PolicyAuthorize() only specifies the new parent, then that authorization may be applied to the duplication of any number of other Objects. If the authorizing entity specifies both a new parent and the duplicated Object, then the authorization only applies to that pairing of Object and new parent.

If either policySession→cpHash or policySession→nameHash has been previously set, the TPM shall return TPM_RC_CPHASH. Otherwise, policySession→nameHash will be set to:

\[\text{nameHash} := H_{\text{policyAlg}}(\text{objectName} || \text{newParentName})\] (30)

NOTE 3
It is allowed that policySession→nameHash and policySession→cpHash share the same memory space.

The policySession→policyDigest will be updated according to the setting of includeObject. If equal to YES, policySession→policyDigest is updated by:

\[\text{policyDigest}_{\text{new}} := H_{\text{policyAlg}}(\text{policyDigest}_{\text{old}} || \text{TPM_CC_PolicyDuplicationSelect} || \text{objectName} || \text{newParentName} || \text{includeObject})\] (31)

If includeObject is NO, policySession→policyDigest is updated by:

\[\text{policyDigest}_{\text{new}} := H_{\text{policyAlg}}(\text{policyDigest}_{\text{old}} || \text{TPM_CC_PolicyDuplicationSelect} || \text{newParentName} || \text{includeObject})\] (32)

NOTE 4
PolicySession→CpHash receives the digest of both Names so that the check performed in TPM2_Duplicate() may be the same regardless of which Names are included in policySession→policyDigest. This means that, when TPM2_PolicyDuplicationSelect() is executed, it is only valid for a specific pair of duplication object and new parent.

If the command succeeds, commandCode in the policy session context is set to TPM_CC_Duplicate.

NOTE 5
The normal use of this command is before a TPM2_PolicyAuthorize(). An authorized entity would approve a policyDigest that allowed duplication to a specific new parent. The authorizing entity may want to limit the authorization so that the approval allows only a specific object to be duplicated to the new parent. In that case, the authorizing entity would approve the policyDigest of equation (31).
### 25.15.2 Command and Response

#### Table 137 — TPM2_PolicyDuplicationSelect Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyDuplicationSelect</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended, Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>objectName</td>
<td>the Name of the object to be duplicated</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>newParentName</td>
<td>the Name of the new parent</td>
</tr>
<tr>
<td>TPMI_YES_NO</td>
<td>includeObject</td>
<td>if YES, the objectName will be included in the value in policySession→policyDigest</td>
</tr>
</tbody>
</table>

#### Table 138 — TPM2_PolicyDuplicationSelect Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.15.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyDuplicationSelect_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_COMMAND_CODE</td>
<td>commandCode of 'policySession'; is not empty</td>
</tr>
<tr>
<td>TPM_RC_CPHASH</td>
<td>cpHash of policySession is not empty</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_PolicyDuplicationSelect(
  PolicyDuplicationSelect_In *in  // IN: input parameter list
) {
  SESSION *session;
  HASH_STATE hashState;
  TPM_CC commandCode = TPM_CC_PolicyDuplicationSelect;
  // Input Validation

  if (session->u1.cpHash.t.size != 0)
    return TPM_RC_CPHASH;

  if (session->commandCode != 0)
    return TPM_RC_COMMAND_CODE;

  // Internal Data Update

  session->u1.cpHash.t.size = CryptStartHash(session->authHashAlg, &hashState);
  CryptUpdateDigest2B(&hashState, &in->objectName.b);

  CryptCompleteHash2B(&hashState, &session->u1.cpHash.b);

  // update policy hash
  // Old policyDigest size should be the same as the new policyDigest size since
  // they are using the same hash algorithm
  session->u2.policyDigest.t.size
    = CryptStartHash(session->authHashAlg, &hashState);

  CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

  CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

  if (in->includeObject == YES)
    CryptUpdateDigest2B(&hashState, &in->objectName.b);
```
54     // add new parent name
55     CryptUpdateDigest2B(&hashState, &in->newParentName.b);
56
57     // add includeObject
58     CryptUpdateDigestInt(&hashState, sizeof(TPMI_YES_NO), &in->includeObject);
59
60     // complete digest
61     CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
62
63     // clear iscpHashDefined bit to indicate now this field contains a nameHash
64     session->attributes.iscpHashDefined = CLEAR;
65
66     // set commandCode in session context
67     session->commandCode = TPM_CC_Duplicate;
68
69     return TPM_RC_SUCCESS;
70  }
25.16 TPM2_PolicyAuthorize

25.16.1 General Description

This command allows policies to change. If a policy were static, then it would be difficult to add users to a policy. This command lets a policy authority sign a new policy so that it may be used in an existing policy.

The authorizing entity signs a structure that contains

\[ aHash := H_{aHashAlg}(approvedPolicy || policyRef) \]  (33)

The \( aHashAlg \) is required to be the \( nameAlg \) of the key used to sign the \( aHash \). The \( aHash \) value is then signed (symmetric or asymmetric) by \( keySign \). That signature is then checked by the TPM in TPM2_VerifySignature() which produces a ticket by

\[ HMAC(proof, (TPM_ST_VERIFIED || aHash || keySign->Name)) \]  (34)

NOTE: The reason for the validation is because of the expectation that the policy will be used multiple times and it is more efficient to check a ticket than to load an object each time to check a signature.

The ticket is then used in TPM2_PolicyAuthorize() to validate the parameters.

The \( keySign \) parameter is required to be a valid object name using \( nameAlg \) other than TPM_ALG_NULL. If the first two octets of \( keySign \) are not a valid hash algorithm, the TPM shall return TPM_RC_HASH. If the remainder of the Name is not the size of the indicated digest, the TPM shall return TPM_RC_SIZE.

The TPM validates that the \( approvedPolicy \) matches the current value of \( policySession->policyDigest \) and if not, shall return TPM_RC_VALUE.

The TPM then validates that the parameters to TPM2_PolicyAuthorize() match the values used to generate the ticket. If so, the TPM will reset \( policySession->policyDigest \) to a Zero Digest. Then it will create a TPM2B_NAME (\( keyName \)) using \( keySign \) and update \( policySession->policyDigest \) with PolicyUpdate() (see 25.2.3).

\[ PolicyUpdate(TPM_CC_PolicyAuthorize, keyName, policyRef) \]  (35)

If the ticket is not valid, the TPM shall return TPM_RC_POLICY.

If \( policySession \) is a trial session, \( policySession->policyDigest \) is extended as if the ticket is valid without actual verification.

NOTE: The unmarshaling process requires that a proper TPMT_TK_VERIFIED be provided for checkTicket but it may be a NULL Ticket.
25.16.2 Command and Response

### Table 139 — TPM2_PolicyAuthorize Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyAuthorize</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td>Auth Index: None</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>approvedPolicy</td>
<td>digest of the policy being approved</td>
</tr>
<tr>
<td>TPM2B_NONCE</td>
<td>policyRef</td>
<td>a policy qualifier</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>keySign</td>
<td>Name of a key that can sign a policy addition</td>
</tr>
<tr>
<td>TPMT_TK_VERIFIED</td>
<td>checkTicket</td>
<td>ticket validating that approvedPolicy and policyRef were signed by keySign</td>
</tr>
</tbody>
</table>

### Table 140 — TPM2_PolicyAuthorize Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.16.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyAuthorize_fp.h"
#include "Policy_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HASH</td>
<td>hash algorithm in keyName is not supported</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>keyName is not the correct size for its hash algorithm</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>the current policyDigest of policySession does not match approvedPolicy, or checkTicket doesn't match the provided values</td>
</tr>
</tbody>
</table>
```

```c
TPM_RC
TPM2_PolicyAuthorize(
    PolicyAuthorize_In   *in  // IN: input parameter list
)
{
    SESSION         *session;
    TPM2B_DIGEST    authHash;
    HASH_STATE      hashState;
    TPMT_TK_VERIFIED ticket;
    TPM_ALG_ID      hashAlg;
    UINT16           digestSize;

    // Input Validation

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Extract from the Name of the key, the algorithm used to compute it's Name
    hashAlg = BYTE_ARRAY_TO_UINT16(in->keySign.t.name);

    // 'keySign' parameter needs to use a supported hash algorithm, otherwise
    // can't tell how large the digest should be
    digestSize = CryptGetHashDigestSize(hashAlg);
    if (digestSize == 0)
        return TPM_RC_HASH + RC_PolicyAuthorize_keySign;

    if (digestSize != (in->keySign.t.size - 2))
        return TPM_RC_SIZE + RC_PolicyAuthorize_keySign;

    // If this is a trial policy, skip all validations
    if (session->attributes.isTrialPolicy == CLEAR)
    {
        // Check that "approvedPolicy" matches the current value of the
        // policyDigest in policy session
        if (!Memory2BEqual(&session->u2.policyDigest.b,
                          &in->approvedPolicy.b))
            return TPM_RC_VALUE + RC_PolicyAuthorize_approvedPolicy;

        // Validate ticket TPMT_TK_VERIFIED
        // Compute aHash. The authorizing object sign a digest
        // aHash := hash(approvedPolicy || policyRef).
        // Start hash
        authHash.t.size = CryptStartHash(hashAlg, &hashState);

        // add approvedPolicy
        CryptUpdateDigest2B(&hashState, &in->approvedPolicy.b);
    }
```
// add policyRef
CryptUpdateDigest2B(&hashState, &in->policyRef.b);

// complete hash
CryptCompleteHash2B(&hashState, &authHash.b);

// re-compute TPMT_TK_VERIFIED
TicketComputeVerified(in->checkTicket.hierarchy, &authHash,
                      &in->keySign, &ticket);

// Compare ticket digest. If not match, return error
if (!Memory2BEqual(&in->checkTicket.digest.b, &ticket.digest.b))
    return TPM_RC_VALUE + RC_PolicyAuthorize_checkTicket;

// Internal Data Update

// Set policyDigest to zero digest
MemorySet(session->u2.policyDigest.t.buffer, 0,
           session->u2.policyDigest.t.size);

// Update policyDigest
PolicyContextUpdate(TPM_CC_PolicyAuthorize, &in->keySign, &in->policyRef,
                    NULL, 0, session);

return TPM_RC_SUCCESS;
25.17 TPM2_PolicyAuthValue

25.17.1 General Description

This command allows a policy to be bound to the authorization value of the authorized object. When this command completes successfully, policySession→isAuthValueNeeded is SET to indicate that the authValue will be included in hmacKey when the authorization HMAC is computed for this session. Additionally, policySession→isPasswordNeeded will be CLEAR.

NOTE If a policy does not use this command, then the hmacKey for the authorized command would only use sessionKey. If sessionKey is not present, then the hmacKey is an Empty Buffer and no HMAC would be computed.

If successful, policySession→policyDigest will be updated with

\[
policyDigest_{\text{new}} := H_{\text{policyAlg}}(policyDigest_{\text{old}} || \text{TPM_CC_PolicyAuthValue})
\]  

(36)
25.17.2 Command and Response

Table 141 — TPM2_PolicyAuthValue Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyAuthValue</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
</tbody>
</table>

Table 142 — TPM2_PolicyAuthValue Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.17.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyAuthValue_fp.h"
#include "Policy_spt_fp.h"

TPM_RC TPM2_PolicyAuthValue(
    PolicyAuthValue_In  *in // IN: input parameter list
)
{
    SESSION             *session;
    TPM_CC               commandCode = TPM_CC_PolicyAuthValue;
    HASH_STATE           hashState;

    // Internal Data Update
    // Get pointer to the session structure
    session = SessionGet(in->policySession);
    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyAuthValue)
    // Start hash
    CryptStartHash(session->authHashAlg, &hashState);
    // add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
    // add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
    // complete the hash and get the results
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
    // update isAuthValueNeeded bit in the session context
    session->attributes.isAuthValueNeeded = SET;
    session->attributes.isPasswordNeeded = CLEAR;
    return TPM_RC_SUCCESS;
}
```
25.18 TPM2_PolicyPassword

25.18.1 General Description

This command allows a policy to be bound to the authorization value of the authorized object. When this command completes successfully, `policySession→isPasswordNeeded` is SET to indicate that `authValue` of the authorized object will be checked when the session is used for authorization. The caller will provide the `authValue` in clear text in the `hmac` parameter of the authorization. The comparison of `hmac` to `authValue` is performed as if the authorization is a password.

NOTE 1 The parameter field in the policy session where the authorization value is provided is called `hmac`. If `TPM2_PolicyPassword()` is part of the sequence, then the field will contain a password and not an HMAC.

If successful, `policySession→policyDigest` will be updated with

\[ policyDigest_{new} \leftarrow H_{policyAlg}(policyDigest_{old} || TPM_CC_PolicyAuthValue) \]  \hspace{1cm} (37)

NOTE 2 This is the same extend value as used with `TPM2_PolicyAuthValue` so that the evaluation may be done using either an HMAC or a password with no change to the `authPolicy` of the object. The reason that two commands are present is to indicate to the TPM if the `hmac` field in the authorization will contain an HMAC or a password value.

When this command is successful, `policySession→isAuthValueNeeded` will be CLEAR.
25.18.2 Command and Response

Table 143 — TPM2_PolicyPassword Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyPassword</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended Auth Index: None</td>
</tr>
</tbody>
</table>

Table 144 — TPM2_PolicyPassword Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.18.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyPassword_fp.h"
#include "Policy_spt_fp.h"

TPM_RC
TPM2_PolicyPassword(
    PolicyPassword_In   *in   // IN: input parameter list
)
{
    SESSION             *session;
    TPM_CC               commandCode = TPM_CC_PolicyAuthValue;
    HASH_STATE           hashState;

    // Internal Data Update

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    // Update policy hash
    // policyDigestnew = hash(policyDigestold || TPM_CC_PolicyAuthValue)
    // Start hash
    CryptStartHash(session->authHashAlg, &hashState);

    // add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);

    // add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);

    // complete the digest
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);

    // Update isPasswordNeeded bit
    session->attributes.isPasswordNeeded = SET;
    session->attributes.isAuthValueNeeded = CLEAR;

    return TPM_RC_SUCCESS;
}
```
25.19 TPM2_PolicyGetDigest

25.19.1 General Description

This command returns the current policyDigest of the session. This command allows the TPM to be used to perform the actions required to pre-compute the authPolicy for an object.
25.19.2 Command and Response

Table 145 — TPM2_PolicyGetDigest Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyGetDigest</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
</tbody>
</table>

Table 146 — TPM2_PolicyGetDigest Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>policyDigest</td>
<td>the current value of the policySession-&gt;policyDigest</td>
</tr>
</tbody>
</table>
25.19.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PolicyGetDigest_fp.h"

TPM_RC TPM2_PolicyGetDigest(
    PolicyGetDigest_In *in, // IN: input parameter list
    PolicyGetDigest_Out *out // OUT: output parameter list
)
{
    SESSION *session;

    // Get pointer to the session structure
    session = SessionGet(in->policySession);

    out->policyDigest = session->u2.policyDigest;

    return TPM_RC_SUCCESS;
}
```
25.20 TPM2_PolicyNvWritten

25.20.1 General Description

This command allows a policy to be bound to the TPMA_NV_WRITTEN attributes. This is a deferred assertion. Values are stored in the policy session context and checked when the policy is used for authorization.

If policySession→checkNVWritten is CLEAR, it is SET and policySession→nvWrittenState is set to writtenSet. If policySession→checkNVWritten is SET, the TPM will return TPM_RC_VALUE if policySession→nvWrittenState and writtenSet are not the same.

If the TPM does not return and error, it will update policySession→policyDigest by

\[
policyDigest_{\text{new}} := H_{\text{policyAlg}}(policyDigest_{\text{old}} || \text{TPM_CC_PolicyNvWritten} || \text{writtenSet})
\]

When the policy session is used to authorize a command, the TPM will fail the command if policySession→checkNVWritten is SET and nvIndex→attributes→TPMA_NV_WRITTEN does not match policySession→nvWrittenState.

NOTE A typical use case is a simple policy for the first write during manufacturing provisioning that would require TPMA_NV_WRITTEN CLEAR and a more complex policy for later use that would require TPMA_NV_WRITTEN SET.
25.20.2 Command and Response

Table 147 — TPM2_PolicyNvWritten Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>Tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PolicyNvWritten</td>
</tr>
<tr>
<td>TPMI_SH_POLICY</td>
<td>policySession</td>
<td>handle for the policy session being extended</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPMI_YES_NO</td>
<td>writtenSet</td>
<td>YES if NV Index is required to have been written</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO if NV Index is required not to have been written</td>
</tr>
</tbody>
</table>

Table 148 — TPM2_PolicyNvWritten Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>Tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
25.20.3 Detailed Actions

Make an NV Index policy dependent on the state of the TPMA_NV_WRITTEN attribute of the index.

### Error Returns

<table>
<thead>
<tr>
<th>Error Return</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>a conflicting request for the attribute has already been processed</td>
</tr>
</tbody>
</table>

```c
#include "InternalRoutines.h"
#include "PolicyNvWritten_fp.h"

TPM_RC TPM2_PolicyNvWritten(
    PolicyNvWritten_In  *in       // IN: input parameter list
) {
    SESSION     *session;
    TPM_CC       commandCode = TPM_CC_PolicyNvWritten;
    HASH_STATE   hashState;

    // Input Validation
    session = SessionGet(in->policySession);
    if (session->attributes.checkNvWritten == SET)
    {
        if ((session->attributes.nvWrittenState == SET)
            != (in->writtenSet == YES))
            return TPM_RC_VALUE + RC_PolicyNvWritten_writtenSet;
    }

    // Internal Data Update
    session->attributes.checkNvWritten = SET;
    session->attributes.nvWrittenState = (in->writtenSet == YES);
    // Update policy hash
    CryptStartHash(session->authHashAlg, &hashState);
    // add old digest
    CryptUpdateDigest2B(&hashState, &session->u2.policyDigest.b);
    // add commandCode
    CryptUpdateDigestInt(&hashState, sizeof(TPM_CC), &commandCode);
    // add the byte of writtenState
    CryptUpdateDigestInt(&hashState, sizeof(TPMI_YES_NO), &in->writtenSet);
    // complete the digest
    CryptCompleteHash2B(&hashState, &session->u2.policyDigest.b);
    return TPM_RC_SUCCESS;
}
```
26  Hierarchy Commands

26.1  TPM2_CreatePrimary

26.1.1  General Description

This command is used to create a Primary Object under one of the Primary Seeds or a Temporary Object under TPM_RH_NULL. The command uses a TPM2B_PUBLIC as a template for the object to be created. The command will create and load a Primary Object. The sensitive area is not returned.

NOTE: Since the sensitive data is not returned, the key cannot be reloaded. It can either be made persistent or it can be recreated.

Any type of object and attributes combination that is allowed by TPM2_Create() may be created by this command. The constraints on templates and parameters are the same as TPM2_Create() except that a Primary Storage Key and a Temporary Storage Key are not constrained to use the algorithms of their parents.

For setting of the attributes of the created object, fixedParent, fixedTPM, userWithAuth, adminWithPolicy, encrypt, and restricted are implied to be SET in the parent (a Permanent Handle). The remaining attributes are implied to be CLEAR.

The TPM will derive the object from the Primary Seed indicated in primaryHandle using an approved KDF. All of the bits of the template are used in the creation of the Primary Key. Methods for creating a Primary Object from a Primary Seed are described in Part 1 of this specification and implemented in Part 4.

If this command is called multiple times with the same inPublic parameter, inSensitive.data, and Primary Seed, the TPM shall produce the same Primary Object.

NOTE  If the Primary Seed is changed, the Primary Objects generated with the new seed shall be statistically unique even if the parameters of the call are the same.

This command requires authorization. Authorization for a Primary Object attached to the Platform Primary Seed (PPS) shall be provided by platformAuth or platformPolicy. Authorization for a Primary Object attached to the Storage Primary Seed (SPS) shall be provided by ownerAuth or ownerPolicy. Authorization for a Primary Key attached to the Endorsement Primary Seed (EPS) shall be provided by endorsementAuth or endorsementPolicy.
### 26.1.2 Command and Response

#### Table 149 — TPM2_CreatePrimary Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_CreatePrimary</td>
</tr>
<tr>
<td>TPMI_RH_HIERARCHY+</td>
<td>@primaryHandle</td>
<td>TPM_RH_ENDORSEMENT, TPM_RH_OWNER,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPM_RH_PLATFORM+(PP), or TPM_RH_NULL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_SENSITIVE_CREATE</td>
<td>inSensitive</td>
<td>the sensitive data, see Part 1 Sensitive Values</td>
</tr>
<tr>
<td>TPM2B_PUBLIC</td>
<td>inPublic</td>
<td>the public template</td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>outsideInfo</td>
<td>data that will be included in the creation data for this object to provide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>permanent, verifiable linkage between this object and some object owner data</td>
</tr>
<tr>
<td>TPML_PCR_SELECTION</td>
<td>creationPCR</td>
<td>PCR that will be used in creation data</td>
</tr>
</tbody>
</table>

#### Table 150 — TPM2_CreatePrimary Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM_HANDLE</td>
<td>objectHandle</td>
<td>Handle for created Primary Object</td>
</tr>
<tr>
<td>TPM2B_PUBLIC</td>
<td>outPublic</td>
<td>the public portion of the created object</td>
</tr>
<tr>
<td>TPM2B_CREATION_DATA</td>
<td>creationData</td>
<td>contains a TPMT_CREATION_DATA</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>creationHash</td>
<td>digest of creationData using nameAlg of outPublic</td>
</tr>
<tr>
<td>TPMT_TK_CREATION</td>
<td>creationTicket</td>
<td>ticket used by TPM2_CertifyCreation() to validate that the creation data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>was produced by the TPM</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
<td>name</td>
<td>the name of the created object</td>
</tr>
</tbody>
</table>
### 26.1.3 Detailed Actions

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td><code>sensitiveDataOrigin</code> is CLEAR when 'sensitive. data' is an Empty Buffer, or is SET when 'sensitive. data' is not empty; 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_KDF</td>
<td>incorrect KDF specified for decrypting keyed hash object</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>there is no free slot for the object</td>
</tr>
<tr>
<td>TPM_RC_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_SIZE</td>
<td>size of public auth policy or sensitive auth value does not match digest size of the name algorithm sensitive data size for the keyed hash object is larger than is allowed for the scheme</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 TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>unknown object type;</td>
</tr>
</tbody>
</table>

```c
#include "InternalRoutines.h"
#include "CreatePrimary_fp.h"
#include "Object_spt_fp.h"
#include <Platform.h>
```

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td><code>sensitiveDataOrigin</code> is CLEAR when 'sensitive. data' is an Empty Buffer, or is SET when 'sensitive. data' is not empty; 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_KDF</td>
<td>incorrect KDF specified for decrypting keyed hash object</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>there is no free slot for the object</td>
</tr>
<tr>
<td>TPM_RC_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_SIZE</td>
<td>size of public auth policy or sensitive auth value does not match digest size of the name algorithm sensitive data size for the keyed hash object is larger than is allowed for the scheme</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 TPM_ALG_NULL</td>
</tr>
<tr>
<td>TPM_RC_TYPE</td>
<td>unknown object type;</td>
</tr>
</tbody>
</table>

```c
TPM2_CreatePrimary(
    CreatePrimary_In    *in,    // IN: input parameter list
    CreatePrimary_Out   *out    // OUT: output parameter list
)
{
    // Local variables
    TPM_RC result = TPM_RC_SUCCESS;
    TPMT_SENSITIVE sensitive;

    // Input Validation
    // The `sensitiveDataOrigin` attribute must be consistent with the setting of // the size of the data object in `inSensitive`.
    if(   (in->primaryHandle.
            objectAttributes.
            sensitiveDataOrigin == SET)
        != (in->inSensitive.
            t.sensitive.
            data.
            t.size == 0 ))
        // Mismatch between the object attributes and the parameter.
        return TPM_RC_ATTRIBUTES + RC_CreatePrimary_inSensitive;

    // Check attributes in input public area. TPM_RC_ATTRIBUTES, TPM_RC_KDF, // TPM_RC_SCHEME, TPM_RC_SIZE, TPM_RC_SYMMETRIC, or TPM_RC_TYPE error may // be returned at this point.
    result = PublicAttributesValidation(FALSE, in->primaryHandle,    // sin->inPublic.t.publicArea);
    if(result != TPM_RC_SUCCESS)
        return RcSafeAddToResult(result, RC_CreatePrimary_inPublic);

    // Validate the sensitive area values
    if( MemoryRemoveTrailingZeros(&in->inSensitive.
            t.sensitive.
            userAuth)
        > CryptGetHashDigestSize(in->inPublic.
            t.publicArea.
            nameAlg))
        return TPM_RC_ATTRIBUTES + RC_CreatePrimary_inSensitive;
}
```
return TPM_RC_SIZE + RC_CreatePrimary_inSensitive;

// Command output

// Generate Primary Object
// The primary key generation process uses the Name of the input public
// template to compute the key. The keys are generated from the template
// before anything in the template is allowed to be changed.
// A TPM_RC_KDF, TPM_RC_SIZE error may be returned at this point
result = CryptCreateObject(in->primaryHandle, &in->inPublic.t.publicArea,
&in->inSensitive.t.sensitive,&sensitive);
if(result != TPM_RC_SUCCESS)
    return result;

// Fill in creation data
FillInCreationData(in->primaryHandle, in->inPublic.t.publicArea.nameAlg,
&in->creationPCR, &in->outsideInfo, &out->creationData,
&out->creationHash);

// Copy public area
out->outPublic = in->inPublic;

// Fill in private area for output
ObjectComputeName(&(out->outPublic.t.publicArea), &out->name);

// Compute creation ticket
TicketComputeCreation(EntityGetHierarchy(in->primaryHandle), &out->name,
    &out->creationHash, &out->creationTicket);

// Create a internal object.  A TPM_RC_OBJECT_MEMORY error may be returned
// at this point.
result = ObjectLoad(in->primaryHandle, &in->inPublic.t.publicArea, &sensitive,
    &out->name, in->primaryHandle, TRUE, &out->objectHandle);
return result;
26.2 TPM2_HierarchyControl

26.2.1 General Description

This command enables and disables use of a hierarchy and its associated NV storage. The command allows phEnable, phEnableNV, shEnable, and ehEnable to be changed when the proper authorization is provided.

This command may be used to CLEAR phEnable and phEnableNV if platformAuth/platformPolicy is provided. phEnable may not be SET using this command.

This command may be used to CLEAR shEnable if either platformAuth/platformPolicy or ownerAuth/ownerPolicy is provided. shEnable may be SET if platformAuth/platformPolicy is provided.

This command may be used to CLEAR ehEnable if either platformAuth/platformPolicy or endorsementAuth/endorsementPolicy is provided. ehEnable may be SET if platformAuth/platformPolicy is provided.

When this command is used to CLEAR phEnable, shEnable, or ehEnable, the TPM will disable use of any persistent entity associated with the disabled hierarchy and will flush any transient objects associated with the disabled hierarchy.

When this command is used to CLEAR shEnable, the TPM will disable access to any NV index that has TPMA_NV_PLATFORMCREATE CLEAR (indicating that the NV Index was defined using ownerAuth). As long as shEnable is CLEAR, the TPM will return an error in response to any command that attempts to operate upon an NV index that has TPMA_NV_PLATFORMCREATE CLEAR.

When this command is used to CLEAR phEnableNV, the TPM will disable access to any NV index that has TPMA_NV_PLATFORMCREATE SET (indicating that the NV Index was defined using platformAuth). As long as phEnableNV is CLEAR, the TPM will return an error in response to any command that attempts to operate upon an NV index that has TPMA_NV_PLATFORMCREATE SET.
26.2.2 Command and Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_HierarchyControl (NV E)</td>
</tr>
<tr>
<td>TPMI_RH_HIERARCHY</td>
<td>@authHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_RH_ENABLES</td>
<td>enable</td>
<td>the enable being modified</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPM_RH_ENDORSEMENT, TPM_RH_OWNER,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPM_RH_PLATFORM, or TPM_RH_PLATFORM_NV</td>
</tr>
<tr>
<td>TPMI_YES_NO</td>
<td>state</td>
<td>YES if the enable should be SET, NO if the enable should be CLEAR</td>
</tr>
</tbody>
</table>

Table 152 — TPM2_HierarchyControl Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
### 26.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "HierarchyControl_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_AUTH_TYPE</td>
<td>authHandle is not applicable to hierarchy in its current state</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_HierarchyControl(
    HierarchyControl_In *in      // IN: input parameter list
) {
    TPM_RC result;
    BOOL select = (in->state == YES);
    BOOL *selected = NULL;

    // Input Validation
    switch(in->enable) {
        // Platform hierarchy has to be disabled by platform auth
        // If the platform hierarchy has already been disabled, only a reboot
        // can enable it again
        case TPM_RH_PLATFORM:
        case TPM_RH_PLATFORM_NV:
            if(in->authHandle != TPM_RH_PLATFORM)
                return TPM_RC_AUTH_TYPE;
            break;

        // ShEnable may be disabled if PlatformAuth/PlatformPolicy or
        // OwnerAuth/OwnerPolicy is provided. If ShEnable is disabled, then it
        // may only be enabled if PlatformAuth/PlatformPolicy is provided.
        case TPM_RH_OWNER:
            if( in->authHandle != TPM_RH_PLATFORM
                && in->authHandle != TPM_RH_OWNER)
                return TPM_RC_AUTH_TYPE;
            if( gc.shEnable == FALSE && in->state == YES
                && in->authHandle != TPM_RH_PLATFORM)
                return TPM_RC_AUTH_TYPE;
            break;

        // EhEnable may be disabled if either PlatformAuth/PlatformPolicy or
        // EndosementAuth/EndorsementPolicy is provided. If EhEnable is disabled,
        // then it may only be enabled if PlatformAuth/PlatformPolicy is
        // provided.
        case TPM_RH_ENDORSEMENT:
            if( in->authHandle != TPM_RH_PLATFORM
                && in->authHandle != TPM_RH_ENDORSEMENT)
                return TPM_RC_AUTH_TYPE;
            if( gc.ehEnable == FALSE && in->state == YES
                && in->authHandle != TPM_RH_PLATFORM)
                return TPM_RC_AUTH_TYPE;
            break;
        default:
            pAssert(FALSE);
            break;
    }

    // Internal Data Update
```
// Enable or disable the selected hierarchy
// Note: the authorization processing for this command may keep these
// command actions from being executed. For example, if phEnable is
// CLEAR, then platformAuth cannot be used for authorization. This
// means that would not be possible to use platformAuth to change the
// state of phEnable from CLEAR to SET.
// If it is decided that platformPolicy can still be used when phEnable
// is CLEAR, then this code could SET phEnable when proper platform
// policy is provided.
switch(in->enable)
{
    case TPM RH_OWNER:
        selected = &gc.shEnable;
        break;
    case TPM RH_ENDORSEMENT:
        selected = &gc.ehEnable;
        break;
    case TPM RH_PLATFORM:
        selected = &g_phEnable;
        break;
    case TPM RH_PLATFORM_NV:
        selected = &gc.phEnableNV;
        break;
    default:
        pAssert(FALSE);
        break;
}
if(selected != NULL && *selected != select)
{
    // Before changing the internal state, make sure that NV is available.
    // Only need to update NV if changing the orderly state
    if(gp.orderlyState != SHUTDOWN_NONE)
    {
        // The command needs NV update. Check if NV is available.
        // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
        // this point
        result = NvIsAvailable();
        if(result != TPM_RC_SUCCESS)
            return result;
    }
    // state is changing and NV is available so modify
    *selected = select;
    // If a hierarchy was just disabled, flush it
    if(select == CLEAR && in->enable != TPM RH_PLATFORM_NV)
    // Flush hierarchy
        ObjectFlushHierarchy(in->enable);
    // orderly state should be cleared because of the update to state clear data
    // This gets processed in ExecuteCommand() on the way out.
    g_clearOrderly = TRUE;
}
return TPM RC_SUCCESS;
26.3 TPM2_SetPrimaryPolicy

26.3.1 General Description

This command allows setting of the authorization policy for the platform hierarchy (*platformPolicy*), the storage hierarchy (*ownerPolicy*), and the endorsement hierarchy (*endorsementPolicy*).

The command requires an authorization session. The session shall use the current *authValue* or satisfy the current *authPolicy* for the referenced hierarchy.

The policy that is changed is the policy associated with *authHandle*.

If the enable associated with *authHandle* is not SET, then the associated authorization values (*authValue* or *authPolicy*) may not be used.
26.3.2 Command and Response

Table 153 — TPM2_SetPrimaryPolicy Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_SetPrimaryPolicy (NV)</td>
</tr>
<tr>
<td>TPMI_RH_HIERARCHY</td>
<td>@authHandle</td>
<td>Auth Index: 1; Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>authPolicy</td>
<td>an authorization policy digest; may be the Empty Buffer</td>
</tr>
<tr>
<td>TPMI_ALG_HASH+</td>
<td>hashAlg</td>
<td>the hash algorithm to use for the policy</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If hashAlg is TPM_ALG_NULL, then this shall be an Empty Buffer.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If the authPolicy is an Empty Buffer, then this field shall be TPM_ALG_NULL.</td>
</tr>
</tbody>
</table>

Table 154 — TPM2_SetPrimaryPolicy Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
# 26.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "SetPrimaryPolicy_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIZE</td>
<td>size of input authPolicy is not consistent with input hash algorithm</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_SetPrimaryPolicy(
    SetPrimaryPolicy_In *in      // IN: input parameter list
) {
    TPM_RC result;

    // Input Validation
    // Check the authPolicy consistent with hash algorithm
    if (in->authPolicy.t.size != 0
        && in->authPolicy.t.size != CryptGetHashDigestSize(in->hashAlg))
        return TPM_RC_SIZE + RC_SetPrimaryPolicy_authPolicy;

    // The command need NV update for OWNER and ENDORSEMENT hierarchy, and
    // might need orderlyState update for PLATFORM hierarchy.
    // Check if NV is available. A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE
    // error may be returned at this point
    result = NvIsAvailable();
    if (result != TPM_RC_SUCCESS)
        return result;

    // Internal Data Update
    // Set hierarchy policy
    switch(in->authHandle) {
    case TPM_RH_OWNER:
        gp.ownerAlg = in->hashAlg;
        gp.ownerPolicy = in->authPolicy;
        NvWriteReserved(NV_OWNER_ALG, &gp.ownerAlg);
        NvWriteReserved(NV_OWNER_POLICY, &gp.ownerPolicy);
        break;
    case TPM_RH_ENDORSEMENT:
        gp.endorsementAlg = in->hashAlg;
        gp.endorsementPolicy = in->authPolicy;
        NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
        NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);
        break;
    case TPM_RH_PLATFORM:
        gc.platformAlg = in->hashAlg;
        gc.platformPolicy = in->authPolicy;
        // need to update orderly state
        g_clearOrderly = TRUE;
        break;
    default:
        pAssert(FALSE);
        break;
    }
    return TPM_RC_SUCCESS;
}
```
26.4 TPM2_ChangePPS

26.4.1 General Description

This replaces the current PPS with a value from the RNG and sets platformPolicy to the default initialization value (the Empty Buffer).

NOTE 1 A policy that is the Empty Buffer can match no policy.

NOTE 2 platformAuth is not changed.

All loaded transient and persistent objects in the Platform hierarchy are flushed.

Saved contexts in the Platform hierarchy that were created under the old PPS will no longer be able to be loaded.

The policy hash algorithm for PCR is reset to TPM_ALG_NULL.

This command does not clear any NV Index values.

NOTE 3 Index values belonging to the Platform are preserved because the indexes may have configuration information that will be the same after the PPS changes. The Platform may remove the indexes that are no longer needed using TPM2_NV_UndefineSpace().

This command requires platformAuth.
26.4.2 Command and Response

Table 155 — TPM2_ChangePPS Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ChangePPS (NV E)</td>
</tr>
<tr>
<td>TPM_RH_PLATFORM</td>
<td>@authHandle</td>
<td>TPM_RH_PLATFORM+{PP}</td>
</tr>
</tbody>
</table>

| Auth Role: USER        | Auth Index: 1 |
|                       |               |
|                       |               |

Table 156 — TPM2_ChangePPS Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
26.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ChangePPS_fp.h"

TPM_RC
TPM2_ChangePPS(
    ChangePPS_In    *in, // IN: input parameter list
)
{
    UINT32         i;
    TPM_RC         result;

    // Check if NV is available. A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE
    // error may be returned at this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Input parameter is not reference in command action
    in = NULL;

    // Internal Data Update
    // Reset platform hierarchy seed from RNG
    CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.PPSeed.t.buffer);
    // Create a new phProof value from RNG to prevent the saved platform
    // hierarchy contexts being loaded
    CryptGenerateRandom(PROOF_SIZE, gp.phProof.t.buffer);
    // Set platform authPolicy to null
    gc.platformAlg = TPM_ALG_NULL;
    gc.platformPolicy.t.size = 0;

    // Flush loaded object in platform hierarchy
    ObjectFlushHierarchy(TPM_RH_PLATFORM);
    // Flush platform evict object and index in NV
    NvFlushHierarchy(TPM_RH_PLATFORM);
    // Save hierarchy changes to NV
    NvWriteReserved(NV_PP_SEED, &gp.PPSeed);
    NvWriteReserved(NV_PH_PROOF, &gp.phProof);

    // Re-initialize PCR policies
    for(i = 0; i < NUM_POLICY_PCR_GROUP; i++)
    {
        gp.pcrPolicies.hashAlg[i] = TPM_ALG_NULL;
        gp.pcrPolicies.policy[i].t.size = 0;
    }
    NvWriteReserved(NV_PCR_POLICIES, &gp.pcrPolicies);

    // orderly state should be cleared because of the update to state clear data
    g_clearOrderly = TRUE;

    return TPM_RC_SUCCESS;
}
```
26.5 TPM2_ChangeEPS

26.5.1 General Description

This replaces the current EPS with a value from the RNG and sets the Endorsement hierarchy controls to their default initialization values: <em>ehEnable</em> is SET, <em>endorsementAuth</em> and <em>endorsementPolicy</em> both equal to the Empty Buffer. It will flush any loaded objects in the EPS hierarchy and not allow objects in the hierarchy associated with the previous EPS to be loaded.

**NOTE** In the reference implementation, <em>ehProof</em> is a non-volatile value from the RNG. It is allowed that the <em>ehProof</em> be generated by a KDF using both the EPS and SPS as inputs. If generated with a KDF, the <em>ehProof</em> can be generated on an as-needed basis or made a non-volatile value.

This command requires <em>platformAuth</em>. 
### 26.5.2 Command and Response

**Table 157 — TPM2\_ChangeEPS Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ChangeEPS (NV E)</td>
</tr>
<tr>
<td>TPM_RH_PLATFORM</td>
<td>@authHandle</td>
<td>Auth Handle: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
</tbody>
</table>

**Table 158 — TPM2\_ChangeEPS Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
26.5.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ChangeEPS_fp.h"

TPM_RC
TPM2_ChangeEPS(
    ChangeEPS_In      *in     // IN: input parameter list
)
{
    TPM_RC          result;
    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;
    // Input parameter is not reference in command action
    in = NULL;

    // Internal Data Update
    // Reset endorsement hierarchy seed from RNG
    CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.EPSeed.t.buffer);
    // Create new ehProof value from RNG
    CryptGenerateRandom(PROOF_SIZE, gp.ehProof.t.buffer);
    // Enable endorsement hierarchy
    gc.ehEnable = TRUE;
    // set authValue buffer to zeros
    MemorySet(gp.endorsementAuth.t.buffer, 0, gp.endorsementAuth.t.size);
    // Set endorsement authValue to null
    gp.endorsementAuth.t.size = 0;
    // Set endorsement authPolicy to null
    gp.endorsementAlg = TPM_ALG_NULL;
    gp.endorsementPolicy.t.size = 0;
    // Flush loaded object in endorsement hierarchy
    ObjectFlushHierarchy(TPM_RH_ENDORSEMENT);
    // Flush evict object of endorsement hierarchy stored in NV
    NvFlushHierarchy(TPM_RH_ENDORSEMENT);
    // Save hierarchy changes to NV
    NvWriteReserved(NV_EP_SEED, &gp.EPSeed);
    NvWriteReserved(NV_EH_PROOF, &gp.ehProof);
    NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
    NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
    NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);
    // orderly state should be cleared because of the update to state clear data
    g_clearOrderly = TRUE;
    return TPM_RC_SUCCESS;
}
```
26.6 TPM2_Clear

26.6.1 General Description

This command removes all TPM context associated with a specific Owner.

The clear operation will:

- flush loaded objects (persistent and volatile) in the Storage and Endorsement hierarchies;
- delete any NV Index with TPMA_NV_PLATFORMCREATE == CLEAR;
- change the SPS to a new value from the TPM’s random number generator (RNG),
- change shProof and ehProof,

NOTE The proof values may be set from the RNG or derived from the associated new Primary Seed. If derived from the Primary Seeds, the derivation of ehProof shall use both the SPS and EPS. The computation shall use the SPS as an HMAC key and the derived value may then be a parameter in a second HMAC in which the EPS is the HMAC key. The reference design uses values from the RNG.

- SET shEnable and ehEnable;
- set ownerAuth, endorsementAuth, and lockoutAuth to the Empty Buffer;
- set ownerPolicy and endorsementPolicy to the Empty Buffer;
- set Clock to zero;
- set resetCount to zero;
- set restartCount to zero; and
- set Safe to YES.

This command requires platformAuth or lockoutAuth. If TPM2_ClearControl() has disabled this command, the TPM shall return TPM_RC_DISABLED.

If this command is authorized using lockoutAuth, the HMAC in the response shall use the new lockoutAuth value (that is, the Empty Buffer) when computing response HMAC.
### 26.6.2 Command and Response

<table>
<thead>
<tr>
<th>Table 159 — TPM2_Clear Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_CC</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>TPMI_RH_CLEAR</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 160 — TPM2_Clear Response</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>TPM_ST</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_RC</td>
</tr>
</tbody>
</table>
#include "InternalRoutines.h"
#include "Clear_fp.h"

## Error Returns

<table>
<thead>
<tr>
<th>Error Return</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_DISABLED</td>
<td>Clear command has been disabled</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_Clear(
    Clear_In *in         // IN: input parameter list
)
{
    TPM_RC              result;
    // Input parameter is not reference in command action
    in = NULL;
    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Input Validation
    if(gp.disableClear) return TPM_RC_DISABLED;

    // Internal Data Update
    // Reset storage hierarchy seed from RNG
    CryptGenerateRandom(PRIMARY_SEED_SIZE, gp.SPSeed.t.buffer);
    // Create new shProof and ehProof value from RNG
    CryptGenerateRandom(PROOF_SIZE, gp.shProof.t.buffer);
    CryptGenerateRandom(PROOF_SIZE, gp.ehProof.t.buffer);
    // Enable storage and endorsement hierarchy
    gc.shEnable = gc.ehEnable = TRUE;
    // set the authValue buffers to zero
    MemorySet(gp.ownerAuth.t.buffer, 0, gp.ownerAuth.t.size);
    MemorySet(gp.endorsementAuth.t.buffer, 0, gp.endorsementAuth.t.size);
    MemorySet(gp.lockoutAuth.t.buffer, 0, gp.lockoutAuth.t.size);
    // Set storage, endorsement and lockout authValue to null
    gp.ownerAuth.t.size = gp.endorsementAuth.t.size = gp.lockoutAuth.t.size = 0;
    // Set storage and endorsement authPolicy to null
    gp.ownerAlg = gp.endorsementAlg = TPM_ALG_NULL;
    gp.ownerPolicy.t.size = gp.endorsementPolicy.t.size = 0;
    // Flush loaded object in storage and endorsement hierarchy
    ObjectFlushHierarchy(TPM_RH_OWNER);
    ObjectFlushHierarchy(TPM_RH_ENDORSEMENT);
    // Flush owner and endorsement object and owner index in NV
    NvFlushHierarchy(TPM_RH_OWNER);
    NvFlushHierarchy(TPM_RH_ENDORSEMENT);
```
// Save hierarchy changes to NV
NvWriteReserved(NV_SP_SEED, &gp.SPSeed);
NvWriteReserved(NV_SH_PROOF, &gp.shProof);
NvWriteReserved(NV_EH_PROOF, &gp.ehProof);
NvWriteReserved(NV_OWNER_AUTH, &gp.ownerAuth);
NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
NvWriteReserved(NV_LOCKOUT_AUTH, &gp.lockoutAuth);
NvWriteReserved(NV_OWNER_ALG, &gp.ownerAlg);
NvWriteReserved(NV_ENDORSEMENT_ALG, &gp.endorsementAlg);
NvWriteReserved(NV_OWNER_POLICY, &gp.ownerPolicy);
NvWriteReserved(NV_ENDORSEMENT_POLICY, &gp.endorsementPolicy);

// Initialize dictionary attack parameters
DAPreInstall_Init();

// Reset clock
go.clock = 0;
go.clockSafe = YES;

// Update the DRBG state whenever writing orderly state to NV
CryptDrbgGetPutState(GET_STATE);
NvWriteReserved(NV_ORDERLY_DATA, &go);

// Reset counters
gp.resetCount = gr.restartCount = gr.clearCount = 0;
gp.auditCounter = 0;
NvWriteReserved(NV_RESET_COUNT, &gp.resetCount);
NvWriteReserved(NV_AUDIT_COUNTER, &gp.auditCounter);

// orderly state should be cleared because of the update to state clear data
g_clearOrderly = TRUE;

return TPM_RC_SUCCESS;
}
26.7  TPM2_ClearControl

26.7.1  General Description

TPM2_ClearControl() disables and enables the execution of TPM2_Clear(). The TPM will SET the TPM’s TPMA_PERMANENT.disableClear attribute if disable is YES and will CLEAR the attribute if disable is NO. When the attribute is SET, TPM2_Clear() may not be executed.

NOTE  This is to simplify the logic of TPM2_Clear(). TPM2_ClearControl() can be called using platformAuth to CLEAR the disableClear attribute and then execute TPM2_Clear().

LockoutAuth may be used to SET disableClear but not to CLEAR it. PlatformAuth may be used to SET or CLEAR disableClear.
### 26.7.2 Command and Response

**Table 161 — TPM2_ClearControl Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ClearControl (NV)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPM_RH_LOCKOUT or TPM_RH_PLATFORM+{PP}</td>
</tr>
<tr>
<td></td>
<td>@auth</td>
<td>Auth Handle: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPML_YES_NO</td>
<td>disable</td>
<td>YES if the disableOwnerClear flag is to be SET, NO if the flag is to be CLEAR.</td>
</tr>
</tbody>
</table>

**Table 162 — TPM2_ClearControl Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
26.7.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ClearControl_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_AUTH_FAIL</td>
<td>authorization is not properly given</td>
</tr>
</tbody>
</table>

TPM_RC

```c
TPM2_ClearControl(
    ClearControl_In *in  // IN: input parameter list
)
{
    TPM_RC      result;
    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Input Validation
    if(in->auth == TPM_RH_LOCKOUT && in->disable == NO)
        return TPM_RC_AUTH_FAIL;

    // Internal Data Update
    if(in->disable == YES)
        gp.disableClear = TRUE;
    else
        gp.disableClear = FALSE;
    // Record the change to NV
    NvWriteReserved(NV_DISABLE_CLEAR, &gp.disableClear);
    return TPM_RC_SUCCESS;
}
```
26.8  TPM2_HierarchyChangeAuth

26.8.1  General Description

This command allows the authorization secret for a hierarchy or lockout to be changed using the current authorization value as the command authorization.

If authHandle is TPM_RH_PLATFORM, then platformAuth is changed. If authHandle is TPM_RH_OWNER, then ownerAuth is changed. If authHandle is TPM_RH_ENDORSEMENT, then endorsementAuth is changed. If authHandle is TPM_RH_LOCKOUT, then lockoutAuth is changed.

If authHandle is TPM_RH_PLATFORM, then Physical Presence may need to be asserted for this command to succeed (see 28.2, "TPM2_PP_Commands").

The authorization value may be no larger than the digest produced by the hash algorithm used for context integrity.

EXAMPLE If SHA384 is used in the computation of the integrity values for saved contexts, then the largest authorization value is 48 octets.
26.8.2 Command and Response

Table 163 — TPM2_HierarchyChangeAuth Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_HierarchyChangeAuth (NV)</td>
</tr>
<tr>
<td>TPMI_RH_HIERARCHY_AUTH</td>
<td>@authHandle</td>
<td>TPM_RH_LOCKOUT, TPM_RH_ENDORSEMENT, TPM_RH_OWNER or TPM_RH_PLATFORM+(PP)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPM2B_AUTH</td>
<td>newAuth</td>
<td>new authorization value</td>
</tr>
</tbody>
</table>

Table 164 — TPM2_HierarchyChangeAuth Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
26.8.3 Detailed Actions

1 #include "InternalRoutines.h"
2 #include "HierarchyChangeAuth_fp.h"
3 #include "Object_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIZE</td>
<td>newAuth size is greater than that of integrity hash digest</td>
</tr>
</tbody>
</table>

4 TPM_RC
5 TPM2_HierarchyChangeAuth(
6    HierarchyChangeAuth_In      *in    // IN: input parameter list
7 )
8 {
9     TPM_RC      result;
10    // The command needs NV update. Check if NV is available.
11    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
12    // this point
13    result = NvIsAvailable();
14    if(result != TPM_RC_SUCCESS) return result;
15    
16    // Make sure the auth value is a reasonable size (not larger than
17    // the size of the digest produced by the integrity hash. The integrity
18    // hash is assumed to produce the longest digest of any hash implemented
19    // on the TPM.
20    if( MemoryRemoveTrailingZeros(&in->newAuth)
21       > CryptGetHashDigestSize(CONTEXT_INTEGRITY_HASH_ALG))
22       return TPM_RC_SIZE + RC_HierarchyChangeAuth_newAuth;
23    
24    // Set hierarchy authValue
25    switch(in->authHandle)
26    {
27        case TPM_RH_OWNER:
28            gp.ownerAuth = in->newAuth;
29            NvWriteReserved(NV_OWNER_AUTH, &gp.ownerAuth);
30            break;
31        case TPM_RH_ENDORSEMENT:
32            gp.endorsementAuth = in->newAuth;
33            NvWriteReserved(NV_ENDORSEMENT_AUTH, &gp.endorsementAuth);
34            break;
35        case TPM_RH_PLATFORM:
36            gc.platformAuth = in->newAuth;
37            // orderly state should be cleared
38            g_clearOrderly = TRUE;
39            break;
40        case TPM_RH_LOCKOUT:
41            gp.lockoutAuth = in->newAuth;
42            NvWriteReserved(NV_LOCKOUT_AUTH, &gp.lockoutAuth);
43            break;
44        default:
45            pAssert(FALSE);
46            break;
47    }
48
49    return TPM_RC_SUCCESS;
50 }
27 Dictionary Attack Functions

27.1 Introduction

A TPM is required to have support for logic that will help prevent a dictionary attack on an authorization value. The protection is provided by a counter that increments when a password authorization or an HMAC authorization fails. When the counter reaches a predefined value, the TPM will not accept, for some time interval, further requests that require authorization and the TPM is in Lockout mode. While the TPM is in Lockout mode, the TPM will return TPM_RC_LOCKED if the command requires use of an object’s or Index’s authValue unless the authorization applies to an entry in the Platform hierarchy.

NOTE Authorizations for objects and NV Index values in the Platform hierarchy are never locked out. However, a command that requires multiple authorizations will not be accepted when the TPM is in Lockout mode unless all of the authorizations reference objects and indexes in the Platform hierarchy.

If the TPM is continuously powered for the duration of newRecoveryTime and no authorization failures occur, the authorization failure counter will be decremented by one. This property is called “self-healing.” Self-healing shall not cause the count of failed attempts to decrement below zero.

The count of failed attempts, the lockout interval, and self-healing interval are settable using TPM2_DictionaryAttackParameters(). The lockout parameters and the current value of the lockout counter can be read with TPM2_GetCapability().

Dictionary attack protection does not apply to an entity associated with a permanent handle (handle type == TPM_HT_PERMANENT).

27.2 TPM2_DictionaryAttackLockReset

27.2.1 General Description

This command cancels the effect of a TPM lockout due to a number of successive authorization failures. If this command is properly authorized, the lockout counter is set to zero.

Only one authorization failure is allowed for this command during a lockoutRecovery interval (set using TPM2_DictionaryAttackParameters()).
27.2.2 Command and Response

Table 165 — TPM2_DictionaryAttackLockReset Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_DictionaryAttackLockReset (NV)</td>
</tr>
<tr>
<td>TPM_RH_LOCKOUT</td>
<td>@lockHandle</td>
<td>TPM_RH_LOCKOUT</td>
</tr>
<tr>
<td>Auth Index: 1</td>
<td></td>
<td>Auth Role: USER</td>
</tr>
</tbody>
</table>

Table 166 — TPM2_DictionaryAttackLockReset Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
27.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "DictionaryAttackLockReset_fp.h"

TPM_RC TPM2_DictionaryAttackLockReset(
    DictionaryAttackLockReset_In     *in     // IN: input parameter list
)
{
    TPM_RC result;

    // Input parameter is not reference in command action
    in = NULL;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Internal Data Update
    gp.failedTries = 0;

    // Record the changes to NV
    NvWriteReserved(NV_FAILED_TRIES, &gp.failedTries);

    return TPM_RC_SUCCESS;
}
```
27.3 TPM2_DictionaryAttackParameters

27.3.1 General Description

This command changes the lockout parameters.
The command requires lockoutAuth.
The timeout parameters (newRecoveryTime and lockoutRecovery) indicate values that are measured with respect to the Time and not Clock.

NOTE Use of Time means that the TPM shall be continuously powered for the duration of a timeout.

If newRecoveryTime is zero, then DA protection is disabled. Authorizations are checked but authorization failures will not cause the TPM to enter lockout.

If newMaxTries is zero, the TPM will be in lockout and use of DA protected entities will be disabled.

If lockoutRecovery is zero, then the recovery interval is a boot cycle (_TPM_Init followed by Startup(CLEAR)).

This command will set the authorization failure count (failedTries) to zero.

Only one authorization failure is allowed for this command during a lockoutRecovery interval.
### 27.3.2 Command and Response

#### Table 167 — TPM2_DictionaryAttackParameters Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_DictionaryAttackParameters (NV)</td>
</tr>
</tbody>
</table>
| TPMI_RH_LOCKOUT              | @lockHandle               | Auth Index: 1  
Auth Role: USER                                                              |
| UINT32                        | newMaxTries               | count of authorization failures before the lockout is imposed              |
| UINT32                        | newRecoveryTime           | time in seconds before the authorization failure count is automatically decremented  
A value of zero indicates that DA protection is disabled.                   |
| UINT32                        | lockoutRecovery           | time in seconds after a lockoutAuth failure before use of lockoutAuth is allowed  
A value of zero indicates that a reboot is required.                         |

#### Table 168 — TPM2_DictionaryAttackParameters Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
27.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "DictionaryAttackParameters_fp.h"

TPM_RC
TPM2_DictionaryAttackParameters(
    DictionaryAttackParameters_In     *in     // IN: input parameter list
)
{
    TPM_RC                  result;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Internal Data Update
    // Set dictionary attack parameters
    gp.maxTries = in->newMaxTries;
    gp.recoveryTime = in->newRecoveryTime;
    gp.lockoutRecovery = in->lockoutRecovery;

    // Set failed tries to 0
    gp.failedTries = 0;

    // Record the changes to NV
    NvWriteReserved(NV_FAILED_TRIES, &gp.failedTries);
    NvWriteReserved(NV_MAX_TRIES, &gp.maxTries);
    NvWriteReserved(NV_RECOVERY_TIME, &gp.recoveryTime);
    NvWriteReserved(NV_LOCKOUT_RECOVERY, &gp.lockoutRecovery);

    return TPM_RC_SUCCESS;
}
```
28 Miscellaneous Management Functions

28.1 Introduction

This clause contains commands that do not logically group with any other commands.

28.2 TPM2_PP_Commands

28.2.1 General Description

This command is used to determine which commands require assertion of Physical Presence (PP) in addition to platformAuth/platformPolicy.

This command requires that auth is TPM_RH_PLATFORM and that Physical Presence be asserted.

After this command executes successfully, the commands listed in setList will be added to the list of commands that require that Physical Presence be asserted when the handle associated with the authorization is TPM_RH_PLATFORM. The commands in clearList will no longer require assertion of Physical Presence in order to authorize a command.

If a command is not in either list, its state is not changed. If a command is in both lists, then it will no longer require Physical Presence (for example, setList is processed first).

Only commands with handle types of TPMI_RH_PLATFORM, TPMI_RH_PROVISION, TPMI_RH_CLEAR, or TPMI_RH_HIERARCHY can be gated with Physical Presence. If any other command is in either list, it is discarded.

When a command requires that Physical Presence be provided, then Physical Presence shall be asserted for either an HMAC or a Policy authorization.

NOTE Physical Presence may be made a requirement of any policy.

TPM2_PP_Commands() always requires assertion of Physical Presence.
28.2.2 Command and Response

Table 169 — TPM2_PP_Commands Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_PP_Commands (NV)</td>
</tr>
<tr>
<td>TPMI_RH_PLATFORM</td>
<td>@auth</td>
<td>Auth Index: 1&lt;br&gt;Auth Role: USER + Physical Presence</td>
</tr>
<tr>
<td>TPML_CC</td>
<td>setList</td>
<td>list of commands to be added to those that will require Physical Presence be asserted</td>
</tr>
<tr>
<td>TPML_CC</td>
<td>clearList</td>
<td>list of commands that will no longer require that Physical Presence be asserted</td>
</tr>
</tbody>
</table>

Table 170 — TPM2_PP_Commands Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
28.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "PP_Commands_fp.h"

TPM_RC
TPM2_PP_Commands(
    PP_Commands_In    *in        // IN: input parameter list
)
{
    UINT32          i;
    TPM_RC      result;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Internal Data Update
    for(i = 0; i < in->setList.count; i++)
        // If command is implemented, set it as PP required. If the input
        // command is not a PP command, it will be ignored at
        // PhysicalPresenceCommandSet().
        if(CommandIsImplemented(in->setList.commandCodes[i]))
            PhysicalPresenceCommandSet(in->setList.commandCodes[i]);

    // Process clear list
    for(i = 0; i < in->clearList.count; i++)
        // If command is implemented, clear it as PP required. If the input
        // command is not a PP command, it will be ignored at
        // PhysicalPresenceCommandClear(). If the input command is
        // TPM2_PP_Commands, it will be ignored as well
        if(CommandIsImplemented(in->clearList.commandCodes[i]))
            PhysicalPresenceCommandClear(in->clearList.commandCodes[i]);

    // Save the change of PP list
    NvWriteReserved(NV_PP_LIST, &gp.ppList);

    return TPM_RC_SUCCESS;
}
```
28.3 TPM2_SetAlgorithmSet

28.3.1 General Description

This command allows the platform to change the set of algorithms that are used by the TPM. The algorithmSet setting is a vendor-dependent value.

If the changing of the algorithm set results in a change of the algorithms of PCR banks, then the TPM will need to be reset (_TPM_Init and TPM2_Startup(TPM_SU_CLEAR)) before the new PCR settings take effect. After this command executes successfully, if startupType in the next TPM2_Startup() is not TPM_SU_CLEAR, the TPM shall return TPM_RC_VALUE and enter Failure mode.

This command does not change the algorithms available to the platform.

NOTE The reference implementation does not have support for this command. In particular, it does not support use of this command to selectively disable algorithms. Proper support would require modification of the unmarshaling code so that each time an algorithm is unmarshaled, it would be verified as being enabled.
28.3.2 Command and Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPML_CC_SetAlgorithmSet (NV)</td>
</tr>
</tbody>
</table>
| TPMI_RH_PLATFORM      | @authHandle | Auth Index: 1  
Auth Role: USER                                                               |
| UINT32                | algorithmSet| a TPM vendor-dependent value indicating the algorithm set selection         |

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
28.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "SetAlgorithmSet_fp.h"

TPM2_SetAlgorithmSet(
    SetAlgorithmSet_In *in) // IN: input parameter list
{
    TPM_RC      result;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Internal Data Update
    gp.algorithmSet = in->algorithmSet;

    // Write the algorithm set changes to NV
    NvWriteReserved(NV_ALGORITHM_SET, &gp.algorithmSet);

    return TPM_RC_SUCCESS;
}
```
29 Field Upgrade

29.1 Introduction

This clause contains the commands for managing field upgrade of the firmware in the TPM. The field upgrade scheme may be used for replacement or augmentation of the firmware installed in the TPM.

EXAMPLE 1 If an algorithm is found to be flawed, a patch of that algorithm might be installed using the firmware upgrade process. The patch might be a replacement of a portion of the code or a complete replacement of the firmware.

EXAMPLE 2 If an additional set of ECC parameters is needed, the firmware process may be used to add the parameters to the TPM data set.

The field upgrade process uses two commands (TPM2_FieldUpgradeStart() and TPM2_FieldUpgradeData()). TPM2_FieldUpgradeStart() validates that a signature on the provided digest is from the TPM manufacturer and that proper authorization is provided using platformPolicy.

NOTE 1 The platformPolicy for field upgraded is defined by the PM and may include requirements that the upgrade be signed by the PM or the TPM owner and include any other constraints that are desired by the PM.

If the proper authorization is given, the TPM will retain the signed digest and enter the Field Upgrade mode (FUM). While in FUM, the TPM will accept TPM2_FieldUpgradeData() commands. It may accept other commands if it is able to complete them using the previously installed firmware. Otherwise, it will return TPM_RC_UPGRADE.

Each block of the field upgrade shall contain the digest of the next block of the field upgrade data. That digest shall be included in the digest of the previous block. The digest of the first block is signed by the TPM manufacturer. That signature and first block digest are the parameters for TPM2_FieldUpgradeStart(). The digest is saved in the TPM as the required digest for the next field upgrade data block and as the identifier of the field upgrade sequence.

For each field upgrade data block that is sent to the TPM by TPM2_FieldUpgradeData(), the TPM shall validate that the digest matches the required digest and if not, shall return TPM_RC_VALUE. The TPM shall extract the digest of the next expected block and return that value to the caller, along with the digest of the first data block of the update sequence.

The system may attempt to abandon the firmware upgrade by using a zero-length buffer in TPM2_FieldUpdateData(). If the TPM is able to resume operation using the firmware present when the upgrade started, then the TPM will indicate that it has abandon the update by setting the digest of the next block to the Empty Buffer. If the TPM cannot abandon the update, it will return the expected next digest.

The system may also attempt to abandon the update because of a power interruption. If the TPM is able to resume normal operations, then it will respond normally to TPM2_Startup(). If the TPM is not able to resume normal operations, then it will respond to any command but TPM2_FieldUpgradeData() with TPM_RC_FIELDUPGRADE.

After a _TPM_Init, system software may not be able to resume the field upgrade that was in process when the power interruption occurred. In such case, the TPM firmware may be reset to one of two other values:

- the original firmware that was installed at the factory (“initial firmware”); or
- the firmware that was in the TPM when the field upgrade process started (“previous firmware”).

The TPM retains the digest of the first block for these firmware images and checks to see if the first block after _TPM_Init matches either of those digests. If so, the firmware update process restarts and the original firmware may be loaded.
NOTE 2 The TPM is required to accept the previous firmware as either a vendor-provided update or as recovered from the TPM using TPM2_FirmwareRead().

When the last block of the firmware upgrade is loaded into the TPM (indicated to the TPM by data in the data block in a TPM vendor-specific manner), the TPM will complete the upgrade process. If the TPM is able to resume normal operations without a reboot, it will set the hash algorithm of the next block to TPM_ALG_NULL and return TPM_RC_SUCCESS. If a reboot is required, the TPM shall return TPM_RC_REBOOT in response to the last TPM2_FieldUpgradeData() and all subsequent TPM commands until a _TPM_Init is received.

NOTE 3 Because no additional data is allowed when the response code is not TPM_RC_SUCCESS, the TPM returns TPM_RC_SUCCESS for all calls to TPM2_FieldUpgradeData() except the last. In this manner, the TPM is able to indicate the digest of the next block. If a _TPM_Init occurs while the TPM is in FUM, the next block may be the digest for the first block of the original firmware. If it is not, then the TPM will not accept the original firmware until the next _TPM_Init when the TPM is in FUM.

During the field upgrade process, the TPM shall preserve:

- Primary Seeds;
- Hierarchy authValue, authPolicy, and proof values;
- Lockout authValue and authorization failure count values;
- PCR authValue and authPolicy values;
- NV Index allocations and contents;
- Persistent object allocations and contents; and
- Clock.
29.2  TPM2_FieldUpgradeStart

29.2.1 General Description

This command uses platformPolicy and a TPM Vendor Authorization Key to authorize a Field Upgrade Manifest.

If the signature checks succeed, the authorization is valid and the TPM will accept TPM2_FieldUpgradeData().

This signature is checked against the loaded key referenced by keyHandle. This key will have a Name that is the same as a value that is part of the TPM firmware data. If the signature is not valid, the TPM shall return TPM_RC_SIGNATURE.

NOTE A loaded key is used rather than a hard-coded key to reduce the amount of memory needed for this key data in case more than one vendor key is needed.
### 29.2.2 Command and Response

#### Table 173 — TPM2_FieldUpgradeStart Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_FieldUpgradeStart&lt;br&gt;TPM_RH_PLATFORM+(PP)&lt;br&gt;Auth Index:1&lt;br&gt;Auth Role: ADMIN</td>
</tr>
<tr>
<td>TPMI_RH_PLATFORM</td>
<td>@authorization</td>
<td>Auth Index:1&lt;br&gt;Auth Role: ADMIN</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>keyHandle</td>
<td>handle of a public area that contains the TPM Vendor Authorization Key that will be used to validate&lt;br&gt;manifestSignature&lt;br&gt;Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_DIGEST</td>
<td>fuDigest</td>
<td>digest of the first block in the field upgrade sequence</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>manifestSignature</td>
<td>signature over fuDigest using the key associated with&lt;br&gt;keyHandle (not optional)</td>
</tr>
</tbody>
</table>

#### Table 174 — TPM2_FieldUpgradeStart Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
29.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "FieldUpgradeStart_fp.h"
#if CC_FieldUpgradeStart == YES
TPM_RC
TPM2_FieldUpgradeStart(
    FieldUpgradeStart_In *in            // IN: input parameter list
)
{
    // Not implemented
    UNUSED_PARAMETER(in);
    return TPM_RC_SUCCESS;
}
#endif
```
29.3 TPM2_FieldUpgradeData

29.3.1 General Description

This command will take the actual field upgrade image to be installed on the TPM. The exact format of fuData is vendor-specific. This command is only possible following a successful TPM2_FieldUpgradeStart(). If the TPM has not received a properly authorized TPM2_FieldUpgradeStart(), then the TPM shall return TPM_RC_FIELDUPGRADE.

The TPM will validate that the digest of fuData matches an expected value. If so, the TPM may buffer or immediately apply the update. If the digest of fuData does not match an expected value, the TPM shall return TPM_RC_VALUE.
29.3.2 Command and Response

Table 175 — TPM2_FieldUpgradeData Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_FieldUpgradeData (NV)</td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>fuData</td>
<td>field upgrade image data</td>
</tr>
</tbody>
</table>

Table 176 — TPM2_FieldUpgradeData Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMT_HA+</td>
<td>nextDigest</td>
<td>tagged digest of the next block TPM_ALG_NULL if field update is complete</td>
</tr>
<tr>
<td>TPMT_HA</td>
<td>firstDigest</td>
<td>tagged digest of the first block of the sequence</td>
</tr>
</tbody>
</table>
29.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "FieldUpgradeData_fp.h"
#if CC_FieldUpgradeData == YES
TPM_RC
TPM2_FieldUpgradeData(
    FieldUpgradeData_In *in, // IN: input parameter list
    FieldUpgradeData_Out *out // OUT: output parameter list
)
{
    UNUSED_PARAMETER(in);
    UNUSED_PARAMETER(out);
    return TPM_RC_SUCCESS;
}
#endif
```
29.4 TPM2_FirmwareRead

29.4.1 General Description

This command is used to read a copy of the current firmware installed in the TPM.

The presumption is that the data will be returned in reverse order so that the last block in the sequence would be the first block given to the TPM in case of a failure recovery. If the TPM2_FirmwareRead sequence completes successfully, then the data provided from the TPM will be sufficient to allow the TPM to recover from an abandoned upgrade of this firmware.

To start the sequence of retrieving the data, the caller sets `sequenceNumber` to zero. When the TPM has returned all the firmware data, the TPM will return the Empty Buffer as `fuData`.

The contents of `fuData` are opaque to the caller.

NOTE 1 The caller should retain the ordering of the update blocks so that the blocks sent to the TPM have the same size and inverse order as the blocks returned by a sequence of calls to this command.

NOTE 2 Support for this command is optional even if the TPM implements TPM2_FieldUpgradeStart() and TPM2_FieldUpgradeData().
29.4.2 Command and Response

Table 177 — TPM2_FirmwareRead Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_FirmwareRead</td>
</tr>
<tr>
<td>UINT32</td>
<td>sequenceNumber</td>
<td>the number of previous calls to this command in this sequence set to 0 on the first call</td>
</tr>
</tbody>
</table>

Table 178 — TPM2_FirmwareRead Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_MAX_BUFFER</td>
<td>fuData</td>
<td>field upgrade image data</td>
</tr>
</tbody>
</table>
29.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "FirmwareRead_fp.h"

TPM_RC
TPM2_FirmwareRead(
    FirmwareRead_In *in,       // IN: input parameter list
    FirmwareRead_Out *out      // OUT: output parameter list
)
{
    // Not implemented
    UNUSED_PARAMETER(in);
    UNUSED_PARAMETER(out);
    return TPM_RC_SUCCESS;
}
```
30  Context Management

30.1  Introduction

Three of the commands in this clause (TPM2_ContextSave(), TPM2_ContextLoad(), and TPM2_FlushContext()) implement the resource management described in the "Context Management" clause in Part 1.

The fourth command in this clause (TPM2_EvictControl()) is used to control the persistence of a loadable objects in TPM memory. Background for this command may be found in the "Owner and Platform Evict Objects" clause in Part 1.

30.2  TPM2_ContextSave

30.2.1  General Description

This command saves a session context, object context, or sequence object context outside the TPM.

No authorization sessions of any type are allowed with this command and tag is required to be TPM_ST_NO_SESSIONS.

NOTE  This preclusion avoids complex issues of dealing with the same session in handle and in the session area. While it might be possible to provide specificity, it would add unnecessary complexity to the TPM and, because this capability would provide no application benefit, use of authorization sessions for audit or encryption is prohibited.

The TPM shall encrypt and integrity protect the context as described in the "Context Protection" clause in Part 1.

See the “Context Data” clause in Part 2 for a description of the context structure in the response.
### 30.2.2 Command and Response

**Table 179 — TPM2_ContextSave Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td>TPM_ST_NO_SESSIONS</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ContextSave</td>
</tr>
<tr>
<td>TPMI_DH_CONTEXT</td>
<td>saveHandle</td>
<td>handle of the resource to save</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
</tbody>
</table>

**Table 180 — TPM2_ContextSave Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPMS_CONTEXT</td>
<td>context</td>
<td></td>
</tr>
</tbody>
</table>
30.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ContextSave_fp.h"
#include "Context_spt_fp.h"
```

<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 for a session context save</td>
</tr>
<tr>
<td>TPM_RC_TOO_MANY_CONTEXTS</td>
<td>no more contexts can be saved as the counter has maxed out</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_ContextSave(
    ContextSave_In *in, // IN: input parameter list
    ContextSave_Out *out // OUT: output parameter list
) {
    TPM_RC          result;
    UINT16          fingerprintSize; // The size of fingerprint in context
    UINT64          contextID = 0; // session context ID
    TPM2B_SYM_KEY   symKey;
    TPM2B_IV        iv;
    TPM2B_DIGEST    integrity;
    UINT16          integritySize;
    BYTE            *buffer;

    // This command may cause the orderlyState to be cleared due to
    // the update of state reset data. If this is the case, check if NV is
    // available first
    if(gp.orderlyState != SHUTDOWN_NONE) {
        // The command needs NV update. Check if NV is available.
        // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
        // this point
        result = NvIsAvailable();
        if(result != TPM_RC_SUCCESS) return result;
    }

    // Initialize output handle. At the end of command action, the output
    // handle of an object will be replaced, while the output handle
    // for a session will be the same as input
    out->context.savedHandle = in->saveHandle;

    // Get the size of fingerprint in context blob. The sequence value in
    // TPMS_CONTEXT structure is used as the fingerprint
    fingerprintSize = sizeof(out->context.sequence);

    // Compute the integrity size at the beginning of context blob
    integritySize = sizeof(integrity.t.size)
        + CryptGetHashDigestSize(CONTEXT_INTEGRITY_HASH_ALG);

    // Perform object or session specific context save
    switch(HandleGetType(in->saveHandle)) {
        case TPM_HT_TRANSIENT:
            {
```
```
54  OBJECT          *object = ObjectGet(in->saveHandle);
55  OBJECT          *outObject =
56     (OBJECT *)(out->context.contextBlob.t.buffer
57       + integritySize + fingerprintSize);
58
59  // Set size of the context data. The contents of context blob is vendor
60  // defined. In this implementation, the size is size of integrity
61  // plus fingerprint plus the whole internal OBJECT structure
62  out->context.contextBlob.t.size = integritySize +
63     fingerprintSize + sizeof(*object);
64
65  // Copy the whole internal OBJECT structure to context blob, leave
66  // the size for fingerprint
67  *outObject = *object;
68
69  // Increment object context ID
70  gr.objectContextID++;
71  // If object context ID overflows, TPM should be put in failure mode
72  if(gr.objectContextID == 0)
73     FAIL(FATAL_ERROR_INTERNAL);
74
75  // Fill in other return values for an object.
76  out->context.sequence = gr.objectContextID;
77  // For regular object, savedHandle is 0x80000000. For sequence object,
78  // savedHandle is 0x80000001. For object with stClear, savedHandle
79  // is 0x80000002
80  if(ObjectIsSequence(object))
81     {
82    out->context.savedHandle = 0x80000001;
83    SequenceDataImportExport(object, outObject, EXPORT_STATE);
84  }
85  else if(object->attributes.stClear == SET)
86     {
87    out->context.savedHandle = 0x80000002;
88  }
89  else
90     {
91    out->context.savedHandle = 0x80000000;
92  }
93
94  // Get object hierarchy
95  out->context.hierarchy = ObjectDataGetHierarchy(object);
96  break;
97}
98 case TPM_HT_HMAC_SESSION:
99     case TPM_HT_POLICY_SESSION:
100     {
101        SESSION         *session = SessionGet(in->saveHandle);
102
103        // Set size of the context data. The contents of context blob is vendor
104        // defined. In this implementation, the size of context blob is the
105        // size of a internal session structure plus the size of
106        // fingerprint plus the size of integrity
107        out->context.contextBlob.t.size = integritySize +
108            fingerprintSize + sizeof(*session);
109
110        // Copy the whole internal SESSION structure to context blob.
111        // Save space for fingerprint at the beginning of the buffer
112        // This is done before anything else so that the actual context
113        // can be reclaimed after this call
114        MemoryCopy(out->context.contextBlob.t.buffer
115            + integritySize + fingerprintSize,
116            session, sizeof(*session),
117            sizeof(SESSCONTEXT));
118  
119  sizeof(out->context.contextBlob.t.buffer) 
120  - integritySize - fingerprintSize); 
121  
122  // Fill in the other return parameters for a session 
123  // Get a context ID and set the session tracking values appropriately 
124  // TPM_RC_CONTEXT_GAP is a possible error. 
125  // SessionContextSave() will flush the in-memory context 
126  // so no additional errors may occur after this call. 
127  result = SessionContextSave(out->context.savedHandle, &contextID); 
128  if(result != TPM_RC_SUCCESS) return result; 
129  
130  // sequence number is the current session contextID 
131  out->context.sequence = contextID; 
132  
133  // use TPM_RH_NULL as hierarchy for session context 
134  out->context.hierarchy = TPM_RH_NULL; 
135  
136  break; 
137  
138  default: 
139  // SaveContext may only take an object handle or a session handle. 
140  // All the other handle type should be filtered out at unmarshal 
141  pAssert(FALSE); 
142  break; 
143  
144  } 
145  
146  // Save fingerprint at the beginning of encrypted area of context blob. 
147  // Reserve the integrity space 
148  MemoryCopy(out->context.contextBlob.t.buffer + integritySize, 
149  &out->context.sequence, sizeof(out->context.sequence), 
150  sizeof(out->context.contextBlob.t.buffer) - integritySize); 
151  
152  // Compute context encryption key 
153  ComputeContextProtectionKey(&out->context, &symKey, &iv); 
154  
155  // Encrypt context blob 
156  CryptSymmetricEncrypt(out->context.contextBlob.t.buffer + integritySize, 
157  CONTEXT_ENCRYPT_ALG, CONTEXT_ENCRYPT_KEY_BITS, 
158  TPM_ALG_CFB, symKey.t.buffer, &iv, 
159  out->context.contextBlob.t.size - integritySize, 
160  out->context.contextBlob.t.buffer + integritySize); 
161  
162  // Compute integrity hash for the object 
163  // In this implementation, the same routine is used for both sessions 
164  // and objects. 
165  ComputeContextIntegrity(&out->context, &integrity); 
166  
167  // add integrity at the beginning of context blob 
168  buffer = out->context.contextBlob.t.buffer; 
169  TPM2B_DIGEST_Marshal(&integrity, &buffer, NULL); 
170  
171  // orderly state should be cleared because of the update of state reset and 
172  // state clear data 
173  g_clearOrderly = TRUE; 
174  
175  return TPM_RC_SUCCESS;
30.3 TPM2_ContextLoad

30.3.1 General Description

This command is used to reload a context that has been saved by TPM2_ContextSave().

No authorization sessions of any type are allowed with this command and tag is required to be TPM_ST_NO_SESSIONS (see note in 30.2.1).

The TPM will return TPM_RC_HIERARCHY if the context is associated with a hierarchy that is disabled.

NOTE Contexts for authorization sessions and for sequence objects belong to the NULL hierarchy which is never disabled.

See the “Context Data” clause in Part 2 for a description of the values in the context parameter.

If the integrity HMAC of the saved context is not valid, the TPM shall return TPM_RC_INTEGRITY.

The TPM shall perform a check on the decrypted context as described in the “Context Confidentiality Protections” clause of Part 1 and enter failure mode if the check fails.
### 30.3.2 Command and Response

#### Table 181 — TPM2_ContextLoad Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td>TPM_ST_NO_SESSIONS</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ContextLoad</td>
</tr>
<tr>
<td>TPMS_CONTEXT</td>
<td>context</td>
<td>the context blob</td>
</tr>
</tbody>
</table>

#### Table 182 — TPM2_ContextLoad Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPML_DH_CONTEXT</td>
<td>loadedHandle</td>
<td>the handle assigned to the resource after it has been successfully loaded</td>
</tr>
</tbody>
</table>
30.3.3 Detailed Actions

#include "InternalRoutines.h"
#include "ContextLoad_fp.h"
#include "Context_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_CONTEXT_GAP</td>
<td>there is only one available slot and this is not the oldest saved session context</td>
</tr>
<tr>
<td>TPM_RC_HANDLE</td>
<td>'context. savedHandle' does not reference a saved session</td>
</tr>
<tr>
<td>TPM_RC_HIERARCHY</td>
<td>'context.hierarchy' is disabled</td>
</tr>
<tr>
<td>TPM_RC_INTEGRITY</td>
<td>context integrity check fail</td>
</tr>
<tr>
<td>TPM_RC_OBJECT_MEMORY</td>
<td>no free slot for an object</td>
</tr>
<tr>
<td>TPM_RC_SESSION_MEMORY</td>
<td>no free session slots</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>incorrect context blob size</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_ContextLoad(
    ContextLoad_In *in, // IN: input parameter list
    ContextLoad_Out *out // OUT: output parameter list
)
{
    // Local Variables
    TPM_RC result = TPM_RC_SUCCESS;
    TPM2B_DIGEST ingerityToCompare;
    TPM2B_DIGEST integrity;
    UINT16 integritySize;
    UINT64 fingerprint;
    BYTE *buffer;
    INT32 size;
    TPM_HT handleType;
    TPM2B_SYM_KEY symKey;
    TPM2B_IV iv;

    // Input Validation
    // Check context blob size
    handleType = HandleGetType(in->context.savedHandle);

    // Check integrity
    // In this implementation, the same routine is used for both sessions
    // and objects.
    integritySize = sizeof(integrity.t.size)
        + CryptGetHashDigestSize(CONTEXT_INTEGRITY_HASH_ALG);

    // Get integrity from context blob
    buffer = in->context.contextBlob.t.buffer;
    size = (INT32) in->context.contextBlob.t.size;
    result = TPM2B_DIGEST.Unmarshal(&integrity, &buffer, &size);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Compute context integrity
    ComputeContextIntegrity(&in->context, &ingerityToCompare);
// Compare integrity
if (!Memory2BEqual(&integrity.b, &ingerityToCompare.b))
    return TPM_RC_INTEGRITY + RC_ContextLoad_context;

// Compute context encryption key
ComputeContextProtectionKey(&in->context, &symKey, &iv);

// Decrypt context data in place
CryptSymmetricDecrypt(in->context.contextBlob.t.buffer + integritySize,
    CONTEXT_ENCRYPT_ALG, CONTEXT_ENCRYPT_KEY_BITS,
    TPM_ALG_CFB, symKey.t.buffer, &iv,
    in->context.contextBlob.t.size - integritySize,
    in->context.contextBlob.t.buffer + integritySize);

// Read the fingerprint value, skip the leading integrity size
MemoryCopy(&fingerprint, in->context.contextBlob.t.buffer + integritySize,
    sizeof(fingerprint), sizeof(fingerprint));
// Check fingerprint. If the check fails, TPM should be put to failure mode
if (fingerprint != in->context.sequence)
    FAIL(FATAL_ERROR_INTERNAL);

// Perform object or session specific input check
switch (handleType)
{
    case TPM_HT_TRANSIENT:
    {
        // Get a pointer to the object in the context blob
        OBJECT *outObject = (OBJECT *)(in->context.contextBlob.t.buffer
            + integritySize + sizeof(fingerprint));

        // Discard any changes to the handle that the TRM might have made
        in->context.savedHandle = TRANSIENT_FIRST;

        // If hierarchy is disabled, no object context can be loaded in this
        // hierarchy
        if (!HierarchyIsEnabled(in->context.hierarchy))
            return TPM_RC_HIERARCHY + RC_ContextLoad_context;

        // Restore object. A TPM_RC_OBJECT_MEMORY error may be returned at
        // this point
        result = ObjectContextLoad(outObject, &out->loadedHandle);
        if (result != TPM_RC_SUCCESS)
            return result;

        // If this is a sequence object, the crypto library may need to
        // reformat the data into an internal format
        if (ObjectIsSequence(outObject))
            SequenceDataImportExport(ObjectGet(out->loadedHandle),
                outObject, IMPORT_STATE);

        break;
    }
    case TPM_HT_POLICY_SESSION:
    case TPM_HT_HMAC_SESSION:
    {
        SESSION *session = (SESSION *)(in->context.contextBlob.t.buffer
            + integritySize + sizeof(fingerprint));

        // This command may cause the orderlyState to be cleared due to
        // the update of state reset data. If this is the case, check if NV is
        // available first
if(gp.orderlyState != SHUTDOWN_NONE)
{
    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned
    // at this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;
}

if(!SessionIsSaved(in->context.savedHandle))
    return TPM_RC_HANDLE + RC_ContextLoad_context;

    // Restore session. A TPM_RC_SESSION_MEMORY, TPM_RC_CONTEXT_GAP error
    // may be returned at this point
    result = SessionContextLoad(session, &in->context.savedHandle);
    if(result != TPM_RC_SUCCESS)
        return result;

    out->loadedHandle = in->context.savedHandle;

    // orderly state should be cleared because of the update of state
    // reset and state clear data
    g_clearOrderly = TRUE;

    break;
}

default:
    // Context blob may only have an object handle or a session handle.
    // All the other handle type should be filtered out at unmarshal
    pAssert(FALSE);
    break;

return TPM_RC_SUCCESS;
30.4 TPM2_FlushContext

30.4.1 General Description

This command causes all context associated with a loaded object or session to be removed from TPM memory.

This command may not be used to remove a persistent object from the TPM.

A session does not have to be loaded in TPM memory to have its context flushed. The saved session context associated with the indicated handle is invalidated.

No sessions of any type are allowed with this command and tag is required to be TPM_ST_NO_SESSIONS (see note in 30.2.1).

If the handle is for a transient object and the handle is not associated with a loaded object, then the TPM shall return TPM_RC_HANDLE.

If the handle is for an authorization session and the handle does not reference a loaded or active session, then the TPM shall return TPM_RC_HANDLE.

NOTE

flushHandle is a parameter and not a handle. If it were in the handle area, the TPM would validate that the context for the referenced entity is in the TPM. When a TPM2_FlushContext references a saved session context, it is not necessary for the context to be in the TPM.
### 30.4.2 Command and Response

#### Table 183 — TPM2_FlushContext Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td>TPM_ST_NO_SESSIONS</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_FlushContext</td>
</tr>
<tr>
<td>TPMI_DH_CONTEXT</td>
<td>flushHandle</td>
<td>the handle of the item to flush</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NOTE This is a use of a handle as a parameter.</td>
</tr>
</tbody>
</table>

#### Table 184 — TPM2_FlushContext Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
30.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "FlushContext_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HANDLE</td>
<td>flushHandle does not reference a loaded object or session</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_FlushContext(
    FlushContext_In *in  // IN: input parameter list
)
{
    // Internal Data Update
    // Call object or session specific routine to flush
    switch(HandleGetType(in->flushHandle))
    {
    case TPM_HT_TRANSIENT:
        if(!ObjectIsPresent(in->flushHandle))
            return TPM_RC_HANDLE;
        // Flush object
        ObjectFlush(in->flushHandle);
        break;
    case TPM_HT_HMAC_SESSION:
        case TPM_HT_POLICY_SESSION:
        if(!SessionIsLoaded(in->flushHandle)
            && !SessionIsSaved(in->flushHandle)
        )
            return TPM_RC_HANDLE;
        // If the session to be flushed is the exclusive audit session, then
        // indicate that there is no exclusive audit session any longer.
        if(in->flushHandle == g_exclusiveAuditSession)
            g_exclusiveAuditSession = TPM_RH_UNASSIGNED;
        // Flush session
        SessionFlush(in->flushHandle);
        break;
        default:
        // This command only take object or session handle. Other handles
        // should be filtered out at handle unmarshal
        pAssert(FALSE);
        break;
    }
    return TPM_RC_SUCCESS;
}
30.5 TPM2_EvictControl

30.5.1 General Description

This command allows a transient object to be made persistent or a persistent object to be evicted.

NOTE 1  A transient object is one that may be removed from TPM memory using either TPM2_FlushContext() or TPM2_Startup(). A persistent object is not removed from TPM memory by TPM2_FlushContext() or TPM2_Startup().

If objectHandle is a transient object, then the call is to make the object persistent and assign persistentHandle to the persistent version of the object. If objectHandle is a persistent object, then the call is to evict the persistent object.

Before execution of TPM2_EvictControl code below, the TPM verifies that objectHandle references an object that is resident on the TPM and that persistentHandle is a valid handle for a persistent object.

NOTE 2  This requirement simplifies the unmarshaling code so that it only need check that persistentHandle is always a persistent object.

If objectHandle references a transient object:

a) The TPM shall return TPM_RC_ATTRIBUTES if
   1) it is in the hierarchy of TPM_RH_NULL,
   2) only the public portion of the object is loaded, or
   3) the stClear is SET in the object or in an ancestor key.

b) The TPM shall return TPM_RC_HIERARCHY if the object is not in the proper hierarchy as determined by auth.
   1) If auth is TPM_RH_PLATFORM, the proper hierarchy is the Platform hierarchy.
   2) If auth is TPM_RH_OWNER, the proper hierarchy is either the Storage or the Endorsement hierarchy.

c) The TPM shall return TPM_RC_RANGE if persistentHandle is not in the proper range as determined by auth.
   1) If auth is TPM_RH_OWNER, then persistentHandle shall be in the inclusive range of 81 00 00 0016 to 81 7F FF FF16.
   2) If auth is TPM_RH_PLATFORM, then persistentHandle shall be in the inclusive range of 81 80 00 0016 to 81 FF FF FF16.

d) The TPM shall return TPM_RC_NV_DEFINED if a persistent object exists with the same handle as persistentHandle.

e) The TPM shall return TPM_RC_NV_SPACE if insufficient space is available to make the object persistent.

f) The TPM shall return TPM_RC_NV_SPACE if execution of this command will prevent the TPM from being able to hold two transient objects of any kind.

NOTE 3  This requirement anticipates that a TPM may be implemented such that all TPM memory is non-volatile and not subject to endurance issues. In such case, there is no movement of an object between memory of different types and it is necessary that the TPM ensure that it is always possible for the management software to move objects to/from TPM memory in order to ensure that the objects required for command execution can be context restored.
g) If the TPM returns TPM_RC_SUCCESS, the object referenced by objectHandle will not be flushed and both objectHandle and persistentHandle may be used to access the object.

If objectHandle references a persistent object:

h) The TPM shall return TPM_RC_RANGE if objectHandle is not in the proper range as determined by auth. If auth is TPM_RC_OWNER, objectHandle shall be in the inclusive range of $81\ 00\ 00\ 00_{16}$ to $81\ 7F\ FF\ FF_{16}$. If auth is TPM_RC_PLATFORM, objectHandle may be any valid persistent object handle.

i) If the TPM returns TPM_RC_SUCCESS, objectHandle will be removed from persistent memory and no longer be accessible.

NOTE 4 The persistent object is not converted to a transient object, as this would prevent the immediate revocation of an object by removing it from persistent memory.
### 30.5.2 Command and Response

#### Table 185 — TPM2_EvictControl Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPMS_CC_EvictControl (NV)</td>
</tr>
<tr>
<td>TPMI_RH_PROVISION</td>
<td>@auth</td>
<td>Auth Handle: 1, Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT</td>
<td>objectHandle</td>
<td>the handle of a loaded object, Auth Index: None</td>
</tr>
<tr>
<td>TPMI_DH_PERSISTENT</td>
<td>persistentHandle</td>
<td>if <code>objectHandle</code> is a transient object handle, then this is the persistent handle for the object. If <code>objectHandle</code> is a persistent object handle, then this shall be the same value as <code>persistentHandle</code>.</td>
</tr>
</tbody>
</table>

#### Table 186 — TPM2_EvictControl Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
### 30.5.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "EvictControl_fp.h"
```

#### Error Returns | Meaning
--- | ---
TPM_RC_ATTRIBUTES | an object with temporary, stClear or publicOnly attribute SET cannot be made persistent
TPM_RC_HIERARCHY | auth cannot authorize the operation in the hierarchy of evictObject
TPM_RC_HANDLE | evictHandle of the persistent object to be evicted is not the same as the persistentHandle argument
TPM_RC_NV_HANDLE | persistentHandle is unavailable
TPM_RC_NV_SPACE | no space in NV to make evictHandle persistent
TPM_RC_RANGE | persistentHandle is not in the range corresponding to the hierarchy of evictObject

```c
TPM_RC
TPM2_EvictControl(
    EvictControl_In   *in     // IN: input parameter list
)
{
    TPM_RC      result;
    OBJECT      *evictObject;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Input Validation
    evictObject = ObjectGet(in->objectHandle);

    // Temporary, stClear or public only objects can not be made persistent
    if(    evictObject->attributes.temporary == SET
         || evictObject->attributes.stClear == SET
         || evictObject->attributes.publicOnly == SET
    )
        return TPM_RC_ATTRIBUTES + RC_EvictControl_objectHandle;

    // If objectHandle refers to a persistent object, it should be the same as input persistentHandle
    if(    evictObject->attributes.evict == SET
         && evictObject->evictHandle != in->persistentHandle
    )
        return TPM_RC_HANDLE + RC_EvictControl_objectHandle;

    // Additional auth validation
    if(in->auth == TPM_RH_PLATFORM)
    {
        // To make persistent
        if(evictObject->attributes.evict == CLEAR)
        {
            // Platform auth can not set evict object in storage or endorsement hierarchy
        }
    }
}
```
if(evictObject->attributes.ppsHierarchy == CLEAR)
    return TPM_RC_HIERARCHY + RC_EvictControl_objectHandle;

// Platform cannot use a handle outside of platform persistent range.
if(!NvIsPlatformPersistentHandle(in->persistentHandle))
    return TPM_RC_RANGE + RC_EvictControl_persistentHandle;
}

// Platform auth can delete any persistent object
}
else if(in->auth == TPM_RH_OWNER)
{
    // Owner auth can not set or clear evict object in platform hierarchy
    if(evictObject->attributes.ppsHierarchy == SET)
        return TPM_RC_HIERARCHY + RC_EvictControl_objectHandle;

    // Owner cannot use a handle outside of owner persistent range.
    if(   evictObject->attributes.evict == CLEAR
        && !NvIsOwnerPersistentHandle(in->persistentHandle)
    )
        return TPM_RC_RANGE + RC_EvictControl_persistentHandle;
    }
else
{
    // Other auth is not allowed in this command and should be filtered out
    // at unmarshal process
    pAssert(FALSE);
}

// Internal Data Update

// Change evict state
if(evictObject->attributes.evict == CLEAR)
{
    // Make object persistent
    // A TPM_RC_NV_HANDLE or TPM_RC_NV_SPACE error may be returned at this
    // point
    result = NvAddEvictObject(in->persistentHandle, evictObject);
    if(result != TPM_RC_SUCCESS) return result;
}
else
{
    // Delete the persistent object in NV
    NvDeleteEntity(evictObject->evictHandle);
}

return TPM_RC_SUCCESS;
31 Clocks and Timers

31.1 TPM2_ReadClock

31.1.1 General Description

This command reads the current TPMS_TIME_INFO structure that contains the current setting of Time, Clock, resetCount, and restartCount.

No authorization sessions of any type are allowed with this command and tag is required to be TPM_ST_NO_SESSIONS.

NOTE: This command is intended to allow the TCB to have access to values that have the potential to be privacy sensitive. The values may be read without authorization because the TCB will not disclose these values. Since they are not signed and cannot be accessed in a command that uses an authorization session, it is not possible for any entity, other than the TCB, to be assured that the values are accurate.
31.1.2 Command and Response

**Table 187 — TPM2_ReadClock Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td>TPM_ST_NO_SESSIONS</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ReadClock</td>
</tr>
</tbody>
</table>

**Table 188 — TPM2_ReadClock Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>returnCode</td>
<td></td>
</tr>
<tr>
<td>TPMS_TIME_INFO</td>
<td>currentTime</td>
<td></td>
</tr>
</tbody>
</table>
31.1.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ReadClock_fp.h"

TPM_RC
TPM2_ReadClock(
    ReadClock_Out *out) // OUT: output parameter list
{
    // Command Output
    out->currentTime.time = g_time;
    TimeFillInfo(&out->currentTime.clockInfo);
    return TPM_RC_SUCCESS;
}
```
31.2 TPM2_ClockSet

31.2.1 General Description

This command is used to advance the value of the TPM’s Clock. The command will fail if newTime is less than the current value of Clock or if the new time is greater than FF FF 00 00 00 00 00 00 00 00 00 00. If both of these checks succeed, Clock is set to newTime. If either of these checks fails, the TPM shall return TPM_RC_VALUE and make no change to Clock.

NOTE This maximum setting would prevent Clock from rolling over to zero for approximately 8,000 years if the Clock update rate was set so that TPM time was passing 33 percent faster than real time. This would still be more than 6,000 years before Clock would roll over to zero. Because Clock will not roll over in the lifetime of the TPM, there is no need for external software to deal with the possibility that Clock may wrap around.

If the value of Clock after the update makes the volatile and non-volatile versions of TPMS_CLOCK_INFO.clock differ by more than the reported update interval, then the TPM shall update the non-volatile version of TPMS_CLOCK_INFO.clock before returning.

This command requires platformAuth or ownerAuth.
### 31.2.2 Command and Response

**Table 189 — TPM2_CLOCKSET Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_CLOCKSET (NV)</td>
</tr>
<tr>
<td>TPMI_RH_PROVISION</td>
<td>@auth</td>
<td>Auth Handle: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>UINT64</td>
<td>newTime</td>
<td>new Clock setting in milliseconds</td>
</tr>
</tbody>
</table>

**Table 190 — TPM2_CLOCKSET Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>returnCode</td>
<td></td>
</tr>
</tbody>
</table>
31.2.3 Detailed Actions

    #include "InternalRoutines.h"
    #include "ClockSet_fp.h"

Read the current TPMS_TIMER_INFO structure settings

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_VALUE</td>
<td>invalid new clock</td>
</tr>
</tbody>
</table>

    TPM_RC
    TPM2_ClockSet(
        ClockSet_In *in  // IN: input parameter list
    )
    {
        #define CLOCK_UPDATE_MASK  ((1ULL << NV_CLOCK_UPDATE_INTERVAL)- 1)
        UINT64      clockNow; // grab the old value
        go.clock = in->newTime; // set the new value
        // Check to see if the update has caused a need for an nvClock update
        if((in->newTime & CLOCK_UPDATE_MASK) > (clockNow & CLOCK_UPDATE_MASK))
        {
            CryptDrbgGetPutState(GET_STATE);
            NvWriteReserved(NV_ORDERLY_DATA, &go);
            // Now the time state is safe
            go.clockSafe = YES;
        }
        return TPM_RC_SUCCESS;
    }

// Input Validation

    if(in->newTime > 0xFFFF000000000000ULL
        || in->newTime < go.clock)
        return TPM_RC_VALUE + RC_ClockSet_newTime;

// Internal Data Update

    clockNow = go.clock; // grab the old value
    go.clock = in->newTime; // set the new value
    // Check to see if the update has caused a need for an nvClock update
    if((in->newTime & CLOCK_UPDATE_MASK) > (clockNow & CLOCK_UPDATE_MASK))
    {
        CryptDrbgGetPutState(GET_STATE);
        NvWriteReserved(NV_ORDERLY_DATA, &go);
        // Now the time state is safe
        go.clockSafe = YES;
    }
31.3 TPM2_ClockRateAdjust

31.3.1 General Description

This command adjusts the rate of advance of Clock and Time to provide a better approximation to real time. The rateAdjust value is relative to the current rate and not the nominal rate of advance.

EXAMPLE 1 If this command had been called three times with rateAdjust = TPM_CLOCK_COARSE_SLOWER and once with rateAdjust = TPM_CLOCK_COARSE_FASTER, the net effect will be as if the command had been called twice with rateAdjust = TPM_CLOCK_COARSE_SLOWER.

The range of adjustment shall be sufficient to allow Clock and Time to advance at real time but no more. If the requested adjustment would make the rate advance faster or slower than the nominal accuracy of the input frequency, the TPM shall return TPM_RC_VALUE.

EXAMPLE 2 If the frequency tolerance of the TPM's input clock is +/-10 percent, then the TPM will return TPM_RC_VALUE if the adjustment would make Clock run more than 10 percent faster or slower than nominal. That is, if the input oscillator were nominally 100 megahertz (MHz), then 1 millisecond (ms) would normally take 100,000 counts. The update Clock should be adjustable so that 1 ms is between 90,000 and 110,000 counts.

The interpretation of “fine” and “coarse” adjustments is implementation-specific.

The nominal rate of advance for Clock and Time shall be accurate to within 15 percent. That is, with no adjustment applied, Clock and Time shall be advanced at a rate within 15 percent of actual time.

NOTE If the adjustments are incorrect, it will be possible to make the difference between advance of Clock/Time and real time to be as much as 1.15 or ~1.33.

Changes to the current Clock update rate adjustment need not be persisted across TPM power cycles.
31.3.2 Command and Response

Table 191 — TPM2_ClockRateAdjust Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_ClockRateAdjust</td>
</tr>
<tr>
<td>TPMI_RH_PROVISION</td>
<td>@auth</td>
<td>Auth Handle: 1</td>
</tr>
<tr>
<td></td>
<td>Auth Role: USER</td>
<td></td>
</tr>
<tr>
<td>TPM_CLOCK_ADJUST</td>
<td>rateAdjust</td>
<td>Adjustment to current Clock update rate</td>
</tr>
</tbody>
</table>

Table 192 — TPM2_ClockRateAdjust Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>returnCode</td>
<td></td>
</tr>
</tbody>
</table>
31.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "ClockRateAdjust_fp.h"

TPM_RC
TPM2_ClockRateAdjust(
    ClockRateAdjust_In      *in  // IN: input parameter list
) {
    // Internal Data Update
    TimeSetAdjustRate(in->rateAdjust);
    return TPM_RC_SUCCESS;
}
```
32 Capability Commands

32.1 Introduction

The TPM has numerous values that indicate the state, capabilities, and properties of the TPM. These values are needed for proper management of the TPM. The TPM2_GetCapability() command is used to access these values.

TPM2_GetCapability() allows reporting of multiple values in a single call. The values are grouped according to type.

NOTE TPM2_TestParms() is used to determine if a TPM supports a particular combination of algorithm parameters.

32.2 TPM2_GetCapability

32.2.1 General Description

This command returns various information regarding the TPM and its current state.

The capability parameter determines the category of data returned. The property parameter selects the first value of the selected category to be returned. If there is no property that corresponds to the value of property, the next higher value is returned, if it exists.

EXAMPLE 1 The list of handles of transient objects currently loaded in the TPM may be read one at a time. On the first read, set the property to TRANSIENT_FIRST and propertyCount to one. If a transient object is present, the lowest numbered handle is returned and moreData will be YES if transient objects with higher handles are loaded. On the subsequent call, use returned handle value plus 1 in order to access the next higher handle.

The propertyCount parameter indicates the number of capabilities in the indicated group that are requested. The TPM will return the number of requested values (propertyCount) or until the last property of the requested type has been returned.

NOTE 1 The type of the capability is determined by a combination of capability and property.

When all of the properties of the requested type have been returned, the moreData parameter in the response will be set to NO. Otherwise, it will be set to YES.

NOTE 2 The moreData parameter will be YES if there are more properties even if the requested number of capabilities has been returned.

The TPM is not required to return more than one value at a time. It is not required to provide the same number of values in response to subsequent requests.

EXAMPLE 2 A TPM may return 4 properties in response to a TPM2_GetCapability(capability = TPM_CAP_TPMPROPERTY, property = TPM_PT_MANUFACTURER, propertyCount = 8) and for a latter request with the same parameters, the TPM may return as few as one and as many as 8 values.

When the TPM is in Failure mode, a TPM is required to allow use of this command for access of the following capabilities:
• TPM_PT_MANUFACTURER
• TPM_PT_VENDOR_STRING_1
• TPM_PT_VENDOR_STRING_2
• TPM_PT_VENDOR_STRING_3
• TPM_PT_VENDOR_STRING_4
• TPM_PT_VENDOR_TPM_TYPE
• TPM_PT_FIRMWARE_VERSION_1
• TPM_PT_FIRMWARE_VERSION_2

NOTE 3 If the vendor string does not require one of these values, the property type does not need to exist.

A vendor may optionally allow the TPM to return other values.

If in Failure mode and a capability is requested that is not available in Failure mode, the TPM shall return no value.

EXAMPLE 3 Assume the TPM is in Failure mode and the TPM only supports reporting of the minimum required set of properties (the limited set to TPML_TAGGED_PCR_PROPERTY values). If a TPM2_GetCapability is received requesting a capability that has a property type value greater than TPM_PT_FIRMWARE_VERSION_2, the TPM will return a zero length list with the moreData parameter set to NO. If the property type is less than TPM_PT_MANUFACTURER, the TPM will return TPM_PT_MANUFACTURER.

In Failure mode, tag is required to be TPM_ST_NO_SESSIONS or the TPM shall return TPM_RC_FAILURE.

The capability categories and the types of the return values are:

<table>
<thead>
<tr>
<th>capability</th>
<th>property</th>
<th>Return Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_CAP_ALGS</td>
<td>TPM_ALG_ID(1)</td>
<td>TPML_ALG_PROPERTY</td>
</tr>
<tr>
<td>TPM_CAP_HANDLES</td>
<td>TPM_HANDLE</td>
<td>TPML_HANDLE</td>
</tr>
<tr>
<td>TPM_CAP_COMMANDS</td>
<td>TPM_CC</td>
<td>TPML_CCA</td>
</tr>
<tr>
<td>TPM_CAP_PP_COMMANDS</td>
<td>TPM_CC</td>
<td>TPML_CC</td>
</tr>
<tr>
<td>TPM_CAP_AUDIT_COMMANDS</td>
<td>TPM_CC</td>
<td>TPML_CC</td>
</tr>
<tr>
<td>TPM_CAP_PCRS</td>
<td>Reserved</td>
<td>TPML_PCR_SELECTION</td>
</tr>
<tr>
<td>TPM_CAP_TPM_PROPERTIES</td>
<td>TPM_PT</td>
<td>TPML_TAGGED_TPM_PROPERTY</td>
</tr>
<tr>
<td>TPM_CAP_PCR_PROPERTIES</td>
<td>TPM_PT_PCR</td>
<td>TPML_TAGGED_PCR_PROPERTY</td>
</tr>
<tr>
<td>TPM_CAP_ECC_CURVE</td>
<td>TPM_ECC_CURVE(1)</td>
<td>TPML_ECC_CURVE</td>
</tr>
<tr>
<td>TPM_CAP_VENDOR_PROPERTY</td>
<td>manufacturer specific</td>
<td>manufacturer-specific values</td>
</tr>
</tbody>
</table>

NOTES:
(1) The TPM_ALG_ID or TPM_ECC_CURVE is cast to a UINT32
• **TPM_CAP_ALGS** – Returns a list of TPMS_ALG_PROPERTIES. Each entry is an algorithm ID and a set of properties of the algorithm.

• **TPM_CAP_HANDLES** – Returns a list of all of the handles within the handle range of the `property` parameter. The range of the returned handles is determined by the handle type (the most-significant octet (MSO) of the `property`). Any of the defined handle types is allowed.

  **EXAMPLE 4** If the MSO of `property` is TPM_HT_NV_INDEX, then the TPM will return a list of NV Index values.

  **EXAMPLE 5** If the MSO of `property` is TPM_HT_PCR, then the TPM will return a list of PCR.

• For this capability, use of TPM_HT_LOADED_SESSION and TPM_HT_SAVED_SESSION is allowed. Requesting handles with a handle type of TPM_HT_LOADED_SESSION will return handles for loaded sessions. The returned handle values will have a handle type of either TPM_HT_HMAC_SESSION or TPM_HT_POLICY_SESSION. If saved sessions are requested, all returned values will have the TPM_HT_HMAC_SESSION handle type because the TPM does not track the session type of saved sessions.

  **NOTE 2** TPM_HT_LOADED_SESSION and TPM_HT_HMAC_SESSION have the same value, as do TPM_HT_SAVED_SESSION and TPM_HT_POLICY_SESSION. It is not possible to request that the TPM return a list of loaded HMAC sessions without including the policy sessions.

• **TPM_CAP_COMMANDS** – Returns a list of the command attributes for all of the commands implemented in the TPM, starting with the TPM_CC indicated by the `property` parameter. If vendor specific commands are implemented, the vendor-specific command attribute with the lowest `commandIndex`, is returned after the non-vendor-specific (base) command.

  **NOTE 4** The type of the `property` parameter is a TPM_CC while the type of the returned list is TPML_CCA.

• **TPM_CAP_PP_COMMANDS** – Returns a list of all of the commands currently requiring Physical Presence for confirmation of platform authorization. The list will start with the TPM_CC indicated by `property`.

• **TPM_CAP_AUDIT_COMMANDS** – Returns a list of all of the commands currently set for command audit.

• **TPM_CAP_PCRS** – Returns the current allocation of PCR in a TPML_PCR_SELECTION. The `property` parameter shall be zero. The TPM will always respond to this command with the full PCR allocation and `moreData` will be NO.

• **TPM_CAP_TPM_PROPERTIES** – Returns a list of tagged properties. The tag is a TPM_PT and the property is a 32-bit value. The properties are returned in groups. Each property group is on a 256-value boundary (that is, the boundary occurs when the TPM_PT is evenly divisible by 256). The TPM will only return values in the same group as the `property` parameter in the command.

• **TPM_CAP_PCR_PROPERTIES** – Returns a list of tagged PCR properties. The tag is a TPM_PT_PCR and the property is a TPMS_PCR_SELECT. The input command property is a TPM_PT_PCR (see Part 2 for PCR properties to be requested) that specifies the first property to be returned. If `propertyCount` is greater than 1, the list of properties begins with that property and proceeds in TPM_PT_PCR sequence.

  **NOTE 5** If the `propertyCount` selects an unimplemented property, the next higher implemented property is returned.

Each item in the list is a TPMS_PCR_SELECT structure that contains a bitmap of all PCR.

  **NOTE 6** A PCR index in all banks (all hash algorithms) has the same properties, so the hash algorithm is not specified here.
• TPM_CAP_TPM_ECC_CURVES – Returns a list of ECC curve identifiers currently available for use in the TPM.

The moreData parameter will have a value of YES if there are more values of the requested type that were not returned.

If no next capability exists, the TPM will return a zero-length list and moreData will have a value of NO.
32.2.2 Command and Response

Table 193 — TPM2_GetCapability Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_GetCapability</td>
</tr>
<tr>
<td>TPM_CAP</td>
<td>capability</td>
<td>group selection; determines the format of the response</td>
</tr>
<tr>
<td>UINT32</td>
<td>property</td>
<td>further definition of information</td>
</tr>
<tr>
<td>UINT32</td>
<td>propertyCount</td>
<td>number of properties of the indicated type to return</td>
</tr>
</tbody>
</table>

Table 194 — TPM2_GetCapability Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPML_YES_NO</td>
<td>moreData</td>
<td>flag to indicate if there are more values of this type</td>
</tr>
<tr>
<td>TPMS_CAPABILITY_DATA</td>
<td>capabilityData</td>
<td>the capability data</td>
</tr>
</tbody>
</table>
32.2.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "GetCapability_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_HANDLE</td>
<td>value of property is in an unsupported handle range for the TPM_CAP_HANDLES capability value</td>
</tr>
<tr>
<td>TPM_RC_VALUE</td>
<td>invalid capability; or property is not 0 for the TPM_CAP_PCRS capability value</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_GetCapability(
    GetCapability_In *in, // IN: input parameter list
    GetCapability_Out *out // OUT: output parameter list
)
{
    // Command Output
    // Set output capability type the same as input type
    out->capabilityData.capability = in->capability;
    switch(in->capability)
    {
    case TPM_CAP_ALGS:
        out->moreData = AlgorithmCapGetImplemented((TPM_ALG_ID) in->property,
            in->propertyCount, &out->capabilityData.data.algorithms);
        break;
    case TPM_CAP_HANDLES:
        switch(HandleGetType((TPM_HANDLE) in->property))
        {
        case TPM_HT_TRANSIENT:
            // Get list of handles of loaded transient objects
            out->moreData = ObjectCapGetLoaded((TPM_HANDLE) in->property,
                in->propertyCount, &out->capabilityData.data.handles);
            break;
        case TPM_HT_PERSISTENT:
            // Get list of handles of persistent objects
            out->moreData = NvCapGetPersistent((TPM_HANDLE) in->property,
                in->propertyCount, &out->capabilityData.data.handles);
            break;
        case TPM_HT_NV_INDEX:
            // Get list of defined NV index
            out->moreData = NvCapGetIndex((TPM_HANDLE) in->property,
                in->propertyCount, &out->capabilityData.data.handles);
            break;
        case TPM_HT_LOADED_SESSION:
            // Get list of handles of loaded sessions
            out->moreData = SessionCapGetLoaded((TPM_HANDLE) in->property,
                in->propertyCount, &out->capabilityData.data.handles);
            break;
        case TPM_HT_ACTIVE_SESSION:
            // Get list of handles of
            out->moreData = SessionCapGetSaved((TPM_HANDLE) in->property,
                in->propertyCount, &out->capabilityData.data.handles);
        }
    }
```
52 break;
53 case TPM_HT_PCR:
54    // Get list of handles of PCR
55    out->moreData = PCRCapGetHandles((TPM_HANDLE) in->property,
56          in->propertyCount,
57          &out->capabilityData.data.handles);
58 break;
59 case TPM_HT_PERMANENT:
60    // Get list of permanent handles
61    out->moreData = PermanentCapGetHandles(
62          (TPM_HANDLE) in->property,
63          in->propertyCount,
64          &out->capabilityData.data.handles);
65 break;
66 default:
67    // Unsupported input handle type
68    return TPM_RC_HANDLE + RC_GetCapability_property;
69 break;
70 }
71 break;
72 case TPM_CAP_COMMANDS:
73    out->moreData = CommandCapGetCCList((TPM_CC) in->property,
74          in->propertyCount,
75          &out->capabilityData.data.command);
76 break;
77 case TPM_CAP_PP_COMMANDS:
78    out->moreData = PhysicalPresenceCapGetCCList((TPM_CC) in->property,
79          in->propertyCount, &out->capabilityData.data.ppCommands);
80 break;
81 case TPM_CAP_AUDIT_COMMANDS:
82    out->moreData = CommandAuditCapGetCCList((TPM_CC) in->property,
83          in->propertyCount, &out->capabilityData.data.auditCommands);
84 break;
85 case TPM_CAP_PCRS:
86    // Input property must be 0
87    if(in->property != 0)
88        return TPM_RC_VALUE + RC_GetCapability_property;
89    out->moreData = PCRCapGetAllocation(in->propertyCount,
90          &out->capabilityData.data.assignedPCR);
91 break;
92 case TPM_CAP_PCR_PROPERTIES:
93    out->moreData = PCRCapGetProperties((TPM_PT_PCR) in->property,
94          in->propertyCount, &out->capabilityData.data.pcrProperties);
95 break;
96 case TPM_CAP_TPM_PROPERTIES:
97    out->moreData = TPMCapGetProperties((TPM_PT) in->property,
98          in->propertyCount, &out->capabilityData.data.tpmProperties);
99 break;
100 #ifdef TPM_ALG_ECC
101    case TPM_CAP_ECC_CURVES:
102        out->moreData = CryptCapGetECCCurve((TPM_ECC_CURVE) in->property,
103              in->propertyCount, &out->capabilityData.data.eccCurves);
104    break;
105 #endif // TPM_ALG_ECC
106 case TPM_CAP_VENDOR_PROPERTY:
107     // vendor property is not implemented
108 default:
109     // Unexpected TPM_CAP value
110     return TPM_RC_VALUE;
111 break;
return TPM_RC_SUCCESS;
}
32.3   TPM2_TestParms

32.3.1   General Description

This command is used to check to see if specific combinations of algorithm parameters are supported. The TPM will unmarshal the provided TPMT_PUBLIC_PARMS. If the parameters unmarshal correctly, then the TPM will return TPM_RC_SUCCESS, indicating that the parameters are valid for the TPM. The TPM will return the appropriate unmarshaling error if a parameter is not valid.
### 32.3.2 Command and Response

**Table 195 — TPM2_TestParms Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_TestParms</td>
</tr>
<tr>
<td>TPMT_PUBLIC_PARMS</td>
<td>parameters</td>
<td>algorithm parameters to be validated</td>
</tr>
</tbody>
</table>

**Table 196 — TPM2_TestParms Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td>TPM_RC</td>
</tr>
</tbody>
</table>
32.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "TestParms_fp.h"

TPM2_TestParms(  
    TestParms_In     *in      // IN: input parameter list 
)  
{
    // Input parameter is not reference in command action
    in = NULL;

    // The parameters are tested at unmarshal process. We do nothing in command
    // action
    return TPM_RC_SUCCESS;
}
```
33 Non-volatile Storage

33.1 Introduction

The NV commands are used to create, update, read, and delete allocations of space in NV memory. Before an Index may be used, it must be defined (TPM2_NV_DefineSpace()).

An Index may be modified if the proper write authorization is provided or read if the proper read authorization is provided. Different controls are available for reading and writing.

An Index may have an Index-specific authValue and authPolicy. The authValue may be used to authorize reading if TPMA_NV_AUTHREAD is SET and writing if TPMA_NV.AUTHWR_WRITE is SET. The authPolicy may be used to authorize reading if TPMA_NV_POLICYREAD is SET and writing if TPMA_NV_POLICYWRITE is SET.

TPMA_NV_PPREAD and TPMA_NV_PPWRITE indicate if reading or writing of the NV Index may be authorized by platformAuth or platformPolicy.

TPMA_NV_OWNERREAD and TPMA_NV_OWNERWRITE indicate if reading or writing of the NV Index may be authorized by ownerAuth or ownerPolicy.

If an operation on an NV index requires authorization, and the authHandle parameter is the handle of an NV Index, then the nvIndex parameter must have the same value or the TPM will return TPM_RC_NV_AUTHORIZATION.

NOTE 1 This check ensures that the authorization that was provided is associated with the NV Index being authorized.

For creating an Index, ownerAuth may not be used if shEnable is CLEAR and platformAuth may not be used if phEnableNV is CLEAR.

If an Index was defined using platformAuth, then that Index is not accessible when phEnableNV is CLEAR. If an Index was defined using ownerAuth, then that Index is not accessible when shEnable is CLEAR.

For read access control, any combination of TPMA_NV_PPREAD, TPMA_NV_OWNERREAD, TPMA_NV_AUTHREAD, or TPMA_NV_POLICYREAD is allowed as long as at least one is SET.

For write access control, any combination of TPMA_NV_PPWRITE, TPMA_NV_OWNERWRITE, TPMA_NV_AUTHWRITE, or TPMA_NV_POLICYWRITE is allowed as long as at least one is SET.

If an Index has been defined and not written, then any operation on the NV Index that requires read authorization will fail (TPM_RC_NV_INITIALIZED). This check may be made before or after other authorization checks but shall be performed before checking the NV Index authValue. An authorization failure due to the NV Index not having been written shall not be logged by the dictionary attack logic.

If TPMA_NV_CLEAR_STCLEAR is SET, then the TPMA_NV_WRITTEN will be CLEAR on each TPM2_Startup(TPM_SU_CLEAR). TPMA_NV_CLEAR_STCLEAR shall not be SET if TPMA_NV_COUNTER is SET.

The code in the “Detailed Actions” clause of each command is written to interface with an implementation-dependent library that allows access to NV memory. The actions assume no specific layout of the structure of the NV data.

Only one NV Index may be directly referenced in a command.

NOTE 2 This means that, if authHandle references an NV Index, then nvIndex will have the same value. However, this does not limit the number of changes that may occur as side effects. For example, any number of NV Indexes might be relocated as a result of deleting or adding a NV Index.
33.2 NV Counters

When an index has the TPMA_NV_COUNTER attribute set, it behaves as a monotonic counter and may only be updated using TPM2_NV_Increment().

When an NV counter is created, the TPM shall initialize the 8-octet counter value with a number that is greater than any count value for any NV counter on the TPM since the time of TPM manufacture.

An NV counter may be defined with the TPMA_NV_ORDERLY attribute to indicate that the NV Index is expected to be modified at a high frequency and that the data is only required to persist when the TPM goes through an orderly shutdown process. The TPM may update the counter value in RAM and occasionally update the non-volatile version of the counter. An orderly shutdown is one occasion to update the non-volatile count. If the difference between the volatile and non-volatile version of the counter becomes as large as MAX_ORDERLY_COUNT, this shall be another occasion for updating the non-volatile count.

Before an NV counter can be used, the TPM shall validate that the count is not less than a previously reported value. If the TPMA_NV_ORDERLY attribute is not SET, or if the TPM experienced an orderly shutdown, then the count is assumed to be correct. If the TPMA_NV_ORDERLY attribute is SET, and the TPM shutdown was not orderly, then the TPM shall OR MAX_ORDERLY_COUNT to the contents of the non-volatile counter and set that as the current count.

NOTE 1 Because the TPM would have updated the NV Index if the difference between the count values was equal to MAX_ORDERLY_COUNT + 1, the highest value that could have been in the NV Index is MAX_ORDERLY_COUNT so it is safe to restore that value.

NOTE 2 The TPM may implement the RAM portion of the counter such that the effective value of the NV counter is the sum of both the volatile and non-volatile parts. If so, then the TPM may initialize the RAM version of the counter to MAX_ORDERLY_COUNT and no update of NV is necessary.

NOTE 3 When a new NV counter is created, the TPM may search all the counters to determine which has the highest value. In this search, the TPM would use the sum of the non-volatile and RAM portions of the counter. The RAM portion of the counter shall be properly initialized to reflect shutdown process (orderly or not) of the TPM.
33.3 TPM2_NV_DefineSpace

33.3.1 General Description

This command defines the attributes of an NV Index and causes the TPM to reserve space to hold the data associated with the NV Index. If a definition already exists at the NV Index, the TPM will return TPM_RC_NV_DEFINED.

The TPM will return TPM_RC_ATTRIBUTES if more than one of TPMA_NV_COUNTER, TPMA_NV_BITS, or TPMA_NV_EXTEND is SET in publicInfo.

NOTE It is not required that any of these three attributes be set.

The TPM shall return TPM_RC_ATTRIBUTES if TPMA_NV_WRITTEN, TPM_NV_READLOCKED, or TPMA_NV_WRITELOCKED is SET.

If TPMA_NV_COUNTER or TPMA_NV_BITS is SET, then publicInfo→dataSize shall be set to eight (8) or the TPM shall return TPM_RC_SIZE.

If TPMA_NV_EXTEND is SET, then publicInfo→dataSize shall match the digest size of the publicInfo.nameAlg or the TPM shall return TPM_RC_SIZE.

If the NV Index is an ordinary Index and publicInfo→dataSize is larger than supported by the TPM implementation then the TPM shall return TPM_RC_SIZE.

NOTE The limit for the data size may vary according to the type of the index. For example, if the index is has TPMA_NV_ORDERLY SET, then the maximum size of an ordinary NV Index may be less than the size of an ordinary NV Index that has TPMA_NV_ORDERLY CLEAR.

At least one of TPMA_NV_PPREAD, TPMA_NV_OWNERREAD, TPMA_NV_AUTHREAD, or TPMA_NV_POLICYREAD shall be SET or the TPM shall return TPM_RC_ATTRIBUTES.

At least one of TPMA_NV_PPWRITE, TPMA_NV_OWNERWRITE, TPMA_NV_AUTHWRITE, or TPMA_NV_POLICYWRITE shall be SET or the TPM shall return TPM_RC_ATTRIBUTES.

If TPMA_NV_CLEAR_STCLEAR is SET, then TPMA_NV_COUNTER shall be CLEAR or the TPM shall return TPM_RC_ATTRIBUTES.

If platformAuth/platformPolicy is used for authorization, then TPMA_NV_PLATFORMCREATE shall be SET in publicInfo. If ownerAuth/ownerPolicy is used for authorization, TPMA_NV_PLATFORMCREATE shall be CLEAR in publicInfo. If TPMA_NV_PLATFORMCREATE is not set correctly for the authorization, the TPM shall return TPM_RC_ATTRIBUTES.

If TPMA_NV_POLICY_DELETE is SET, then the authorization shall be with platformAuth or the TPM shall return TPM_RC_ATTRIBUTES.

If the implementation does not support TPM2_NV_Increment(), the TPM shall return TPM_RC_ATTRIBUTES if TPMA_NV_COUNTER is SET.

If the implementation does not support TPM2_NV_SetBits(), the TPM shall return TPM_RC_ATTRIBUTES if TPMA_NV_BITS is SET.

If the implementation does not support TPM2_NV_Extend(), the TPM shall return TPM_RC_ATTRIBUTES if TPMA_NV_EXTEND is SET.

If the implementation does not support TPM2_NV_UndefineSpaceSpecial(), the TPM shall return TPM_RC_ATTRIBUTES if TPMA_NV_POLICY_DELETE is SET.

After the successful completion of this command, the NV Index exists but TPMA_NV_WRITTEN will be CLEAR. Any access of the NV data will return TPM_RC_NV_UNINITIALIZED.
In some implementations, an NV Index with the TPMA_NV_COUNTER attribute may require special TPM resources that provide higher endurance than regular NV. For those implementations, if this command fails because of lack of resources, the TPM will return TPM_RC_NV_SPACE.

The value of auth is saved in the created structure. The size of auth is limited to be no larger than the size of the digest produced by the NV Index's nameAlg (TPM_RC_SIZE).
### 33.3.2 Command and Response

**Table 197 — TPM2_NV_DefineSpace Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_DefineSpace (NV)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPML_RH_PROVISION AuthHandle: 1 Auth Role: USER</td>
</tr>
<tr>
<td></td>
<td>@authHandle</td>
<td></td>
</tr>
<tr>
<td>TPM2B_AUTH</td>
<td>auth</td>
<td>the authorization value</td>
</tr>
<tr>
<td>TPM2B_NV_PUBLIC</td>
<td>publicInfo</td>
<td>the public parameters of the NV area</td>
</tr>
</tbody>
</table>

**Table 198 — TPM2_NV_DefineSpace Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.3.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_DefineSpace_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_ATTRIBUTES</td>
<td>attributes of the index are not consistent</td>
</tr>
<tr>
<td>TPM_RC_NV_DEFINED</td>
<td>index already exists</td>
</tr>
<tr>
<td>TPM_RC_HIERARCHY</td>
<td>for authorizations using TPM_RH_PLATFORM phEnable_NV is clear.</td>
</tr>
<tr>
<td>TPM_RC_NV_SPACE</td>
<td>Insufficient space for the index</td>
</tr>
<tr>
<td>TPM_RC_SIZE</td>
<td>'auth-&gt;size' or 'publicInfo-&gt;authPolicy.size' is larger than the digest size of 'publicInfo-&gt;nameAlg', or 'publicInfo-&gt;dataSize' is not consistent with 'publicInfo-&gt;attributes'.</td>
</tr>
</tbody>
</table>

```

`TPM_RC_NV_DefineSpace`

```c
TPM2_NV_DefineSpace(
    NV_DefineSpace_In       *in     // IN: input parameter list
) {
    TPM_RC                   result;
    TPMA_NV                  attributes;
    UINT16                   nameSize;

    nameSize = CryptGetHashDigestSize(in->publicInfo.t.nvPublic.nameAlg);

    // Check if NV is available. NvIsAvailable may return TPM_RC_NV_UNAVAILABLE
    // TPM_RC_NV_RATE or TPM_RC_SUCCESS.
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

    // Input Validation
    // If an index is being created by the owner and shEnable is
    // clear, then we would not reach this point because ownerAuth
    // can't be given when shEnable is CLEAR. However, if phEnable
    // is SET but phEnableNV is CLEAR, we have to check here
    if((in->authHandle == TPM_RH_PLATFORM && gc.phEnableNV == CLEAR)
        return TPM_RC_HIERARCHY + RC_NV_DefineSpace_authHandle;
    attributes = in->publicInfo.t.nvPublic.attributes;

    //TPMS_NV_PUBLIC validation.
    // Counters and bit fields must have a size of 8
    if((attributes.TPMA_NV_COUNTER == SET || attributes.TPMA_NV_BITS == SET)
        && (in->publicInfo.t.nvPublic.dataSize != 8))
        return TPM_RC_SIZE + RC_NV_DefineSpace_publicInfo;

    // check that the authPolicy consistent with hash algorithm
    if((in->publicInfo.t.nvPublic.authPolicy.t.size != 0
        && in->publicInfo.t.nvPublic.authPolicy.t.size != nameSize)
        return TPM_RC_SIZE + RC_NV_DefineSpace_publicInfo;

    // make sure that the authValue is not too large
    MemoryRemoveTrailingZeros(in->auth);
    if(in->auth.t.size > nameSize)
        return TPM_RC_SIZE + RC_NV_DefineSpace_auth;
```

Family “2.0” Published
Level 00 Revision 00.99 Copyright © TCG 2006-2013
Page 397 October 31, 2013
//TPMA_NV validation.
// Locks may not be SET and written cannot be SET
if (attributes.TPMA_NV_WRITTEN == SET
    || attributes.TPMA_NV_WRITELOCKED == SET
    || attributes.TPMA_NV_READLOCKED == SET)
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;

// There must be a way to read the index
if (attributes.TPMA_NV_OWNERREAD == CLEAR
    && attributes.TPMA_NV_PPREAD == CLEAR
    && attributes.TPMA_NV_AUTHREAD == CLEAR
    && attributes.TPMA_NV_POLICYREAD == CLEAR)
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;

// There must be a way to write the index
if (attributes.TPMA_NV_OWNERWRITE == CLEAR
    && attributes.TPMA_NV_PPWRITE == CLEAR
    && attributes.TPMA_NV_AUTHWRITE == CLEAR
    && attributes.TPMA_NV_POLICYWRITE == CLEAR)
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;

// Make sure that no attribute is used that is not supported by the proper command
#if CC_NV_Increment == NO
    if (attributes.TPMA_NV_COUNTER == SET)
        return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
#endif
#if CC_NV_SetBits == NO
    if (attributes.TPMA_NV_BITS == SET)
        return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
#endif
#if CC_NV_Extend == NO
    if (attributes.TPMA_NV_EXTEND == SET)
        return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
#endif
#if CC_NV_UndefineSpaceSpecial == NO
    if (attributes.TPMA_NV_POLICY_DELETE == SET)
        return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
#endif

// Can be COUNTER or BITS or EXTEND but not more than one
if (attributes.TPMA_NV_COUNTER == SET
    && attributes.TPMA_NV_BITS == SET)
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
if (attributes.TPMA_NV_COUNTER == SET
    && attributes.TPMA_NV_EXTEND == SET)
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;
if (attributes.TPMA_NV_BITS == SET
    && attributes.TPMA_NV_EXTEND == SET)
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;

// An index with TPMA_NV_CLEAR_STCLEAR can't be a counter
if (attributes.TPMA_NV_CLEAR_STCLEAR == SET
    && attributes.TPMA_NV_COUNTER == SET)
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;

// The index is allowed to have one of GLOBALLOCK or WRITEDEFINE SET
if (attributes.TPMA_NV_GLOBALLOCK == SET
    && attributes.TPMA_NV_WRITEDEFINE == SET)
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;

// Make sure that the creator of the index can delete the index
if( (in->publicInfo.t.nvPublic.attributes.TPMA_NV_PLATFORMCREATE == SET
    && in->authHandle == TPM_RH_OWNER
    ) || (in->publicInfo.t.nvPublic.attributes.TPMA_NV_PLATFORMCREATE == CLEAR
    && in->authHandle == TPM_RH_PLATFORM
    ) )
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_authHandle;

// If TPMA_NV_POLICY_DELETE is SET, then the index must be defined by
// the platform
if( (in->publicInfo.t.nvPublic.attributes.TPMA_NV_POLICY_DELETE == SET
    && TPM_RH_PLATFORM != in->authHandle
    ) )
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;

// If the NV index is used as a PCR, the data size must match the digest
// size
if( (in->publicInfo.t.nvPublic.attributes.TPMA_NV_EXTEND == SET
    && in->publicInfo.t.nvPublic.dataSize != nameSize
    ) )
    return TPM_RC_ATTRIBUTES + RC_NV_DefineSpace_publicInfo;

// See if the index is already defined.
if(NvIsUndefinedIndex(in->publicInfo.t.nvPublic.nvIndex))
    return TPM_RC_NV_DEFINED;

// Internal Data Update
// define the space.  A TPM_RC_NV_SPACE error may be returned at this point
result = NvDefineIndex(in->publicInfo.t.nvPublic, in->auth);
if(result != TPM_RC_SUCCESS)
    return result;

return TPM_RC_SUCCESS;
33.4 TPM2_NV_UndefineSpace

33.4.1 General Description

This command removes an Index from the TPM.

If `nvIndex` is not defined, the TPM shall return TPM_RC_HANDLE.

If `nvIndex` references an Index that has its TPMA_NV_PLATFORMCREATE attribute SET, the TPM shall return TPM_RC_NV_AUTHORITY unless `platformAuth` is provided.

NOTE An Index with TPMA_NV_PLATFORMCREATE CLEAR may be deleted with `platformAuth` as long as shEnable is SET. If shEnable is CLEAR, indexes created using ownerAuth are not accessible even for deletion by the platform.
33.4.2 Command and Response

Table 199 — TPM2_NV_UndefineSpace Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_UndefineSpace (NV)</td>
</tr>
<tr>
<td>TPMI_RH_PROVISION</td>
<td>@authHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>the NV Index to remove from NV space</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
</tbody>
</table>

Table 200 — TPM2_NV_UndefineSpace Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.4.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_UndefineSpace_fp.h"

Error Returns  Meaning
TPM_RC_ATTRIBUTES  TPMA_NV_POLICY_DELETE is SET in the Index referenced by
                      nvIndex so this command may not be used to delete this Index (see
                      TPM2_NV_UndefineSpaceSpecial())
TPM_RC_NV_AUTHORIZATION  attempt to use ownerAuth to delete an index created by the platform

TPM_RC
TPM2_NV_UndefineSpace(
    NV_UndefineSpace_In *in      // IN: input parameter list
) 
{
    TPM_RC          result;
    NV_INDEX        nvIndex;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Input Validation
    NvGetIndexInfo(in->nvIndex, &nvIndex);

    // This command can't be used to delete an index with TPMA_NV_POLICY_DELETE SET
    if(SET == nvIndex.publicArea.attributes.TPMA_NV_POLICY_DELETE)
        return TPM_RC_ATTRIBUTES + RC_NV_UndefineSpace_nvIndex;

    // The owner may only delete an index that was defined with ownerAuth. The
    // platform may delete an index that was created with either auth.
    if( in->authHandle == TPM_RH_OWNER
        && nvIndex.publicArea.attributes.TPMA_NV_PLATFORMCREATE == SET)
        return TPM_RC_NV_AUTHORIZATION;

    // Internal Data Update
    NvDeleteEntity(in->nvIndex);
    return TPM_RC_SUCCESS;
}```
33.5 TPM2_NV_UndefineSpaceSpecial

33.5.1 General Description

This command allows removal of a platform-created NV Index that has TPMA_NV_POLICY_DELETE SET.

This command requires that the policy of the NV Index be satisfied before the NV Index may be deleted. Because administrative role is required, the policy must contain a command that sets the policy command code to TPM_CC_NV_UndefineSpaceSpecial. This indicates that the policy that is being used is a policy that is for this command, and not a policy that would approve another use. That is, authority to use an object does not grant authority to undefined the object.

If nvIndex is not defined, the TPM shall return TPM_RC_HANDLE.

If nvIndex references an Index that has its TPMA_NV_PLATFORMCREATE or TPMA_NV_POLICY_DELETE attribute CLEAR, the TPM shall return TPM_RC_NV_ATTRIBUTES.

NOTE An Index with TPMA_NV_PLATFORMCREATE CLEAR may be deleted with TPM2_UndefineSpace() as long as shEnable is SET. If shEnable is CLEAR, indexes created using ownerAuth are not accessible even for deletion by the platform.
33.5.2 Command and Response

Table 201 — TPM2_NV_UndefineSpaceSpecial Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_UndefineSpaceSpecial {NV}</td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>@nvIndex</td>
<td>Index to be deleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: ADMIN</td>
</tr>
<tr>
<td>TPMI_RH_PLATFORM</td>
<td>@platform</td>
<td>TPM_RH_PLATFORM + {PP}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
</tbody>
</table>

Table 202 — TPM2_NV_UndefineSpaceSpecial Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.5.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_UndefineSpaceSpecial_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>TPMA_NV_POLICY_DELETE is not SET in the Index referenced by nvIndex</td>
</tr>
</tbody>
</table>

TPM_RC
TPM2_NV_UndefineSpaceSpecial(
    NV_UndefineSpaceSpecial_In *in // IN: input parameter list
)
{
    TPM_RC result;
    NV_INDEX nvIndex;
    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

    // Input Validation
    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);
    // This operation only applies when the TPMA_NV_POLICY_DELETE attribute is SET
    if(CLEAR == nvIndex.publicArea.attributes.TPMA_NV_POLICY_DELETE)
        return TPM_RC_ATTRIBUTES + RC_NV_UndefineSpaceSpecial_nvIndex;

    // Internal Data Update
    // Call implementation dependent internal routine to delete NV index
    NvDeleteEntity(in->nvIndex);
    return TPM_RC_SUCCESS;
}
```
33.6 TPM2_NV_ReadPublic

33.6.1 General Description

This command is used to read the public area and Name of an NV Index. The public area of an Index is not privacy-sensitive and no authorization is required to read this data.
### 33.6.2 Command and Response

<table>
<thead>
<tr>
<th>Table 203 — TPM2_NV_ReadPublic Command</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_CC</td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 204 — TPM2_NV_ReadPublic Response</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>TPM_ST</td>
</tr>
<tr>
<td>UINT32</td>
</tr>
<tr>
<td>TPM_RC</td>
</tr>
<tr>
<td>TPM2B_NV_PUBLIC</td>
</tr>
<tr>
<td>TPM2B_NAME</td>
</tr>
</tbody>
</table>
33.6.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_ReadPublic_fp.h"

TPM_RC
TPM2_NV_ReadPublic(
    NV_ReadPublic_In *in, // IN: input parameter list
    NV_ReadPublic_Out *out // OUT: output parameter list
)
{
    NV_INDEX        nvIndex;

    // Command Output
    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);
    // Copy data to output
    out->nvPublic.t.nvPublic = nvIndex.publicArea;
    // Compute NV name
    out->nvName.t.size = NvGetName(in->nvIndex, &out->nvName.t.name);
    return TPM_RC_SUCCESS;
}
```
33.7  TPM2_NV_Write

33.7.1  General Description

This command writes a value to an area in NV memory that was previously defined by TPM2_NV_DefineSpace().

Proper authorizations are required for this command as determined by TPMA_NV_PPWRITE; TPMA_NV_OWNERWRITE; TPMA_NV_AUTHWRITE; and, if TPMA_NV_POLICY_WRITE is SET, the authPolicy of the NV Index.

If the TPMA_NV_WRITELOCKED attribute of the NV Index is SET, then the TPM shall return TPM_RC_NV_LOCKED.

NOTE 1  If authorization sessions are present, they are checked before checks to see if writes to the NV Index are locked.

If TPMA_NV_COUNTER, TPMA_NV_BITS or TPMA_NV_EXTEND of the NV Index is SET, then the TPM shall return TPM_RC_NV_ATTRIBUTE.

If the size of the data parameter plus the offset parameter adds to a value that is greater than the size of the NV Index data, the TPM shall return TPM_RC_NV_RANGE and not write any data to the NV Index.

If the TPMA_NV_WRITEALL attribute of the NV Index is SET, then the TPM shall return TPM_RC_NV_RANGE if the size of the data parameter of the command is not the same as the data field of the NV Index.

If all checks succeed, the TPM will merge the data.size octets of data.buffer value into the nvIndex→data starting at nvIndex→data[offset]. If the NV memory is implemented with a technology that has endurance limitations, the TPM shall check that the merged data is different from the current contents of the NV Index and only perform a write to NV memory if they differ.

After successful completion of this command, TPMA_NV_WRITTEN for the NV Index will be SET.

NOTE 2  Once SET, TPMA_NV_WRITTEN remains SET until the NV Index is undefined or the NV Index is cleared.
33.7.2 Command and Response

Table 205 — TPM2_NV_Write Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_Write (NV)</td>
</tr>
<tr>
<td>TPMI_RH_NV_AUTH</td>
<td>@authHandle</td>
<td>handle indicating the source of the authorization value</td>
</tr>
<tr>
<td></td>
<td>Auth Index: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Auth Role: USER</td>
<td></td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>the NV Index of the area to write</td>
</tr>
<tr>
<td></td>
<td>Auth Index: None</td>
<td></td>
</tr>
<tr>
<td>TPM2B_MAX_NV_BUFFER</td>
<td>data</td>
<td>the data to write</td>
</tr>
<tr>
<td>UINT16</td>
<td>offset</td>
<td>the offset into the NV Area</td>
</tr>
</tbody>
</table>

Table 206 — TPM2_NV_Write Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.7.3 Detailed Actions

```
#include "InternalRoutines.h"
#include "NV_Write_fp.h"
#include "NV_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>Index referenced by nvIndex has either TPMA_NV_BITS, TPMA_NV_COUNTER, or TPMA_NV_EVENT attribute SET</td>
</tr>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>the authorization was valid but the authorizing entity (authHandle) is not allowed to write to the Index referenced by nvIndex</td>
</tr>
<tr>
<td>TPM_RC_NV_LOCKED</td>
<td>Index referenced by nvIndex is write locked</td>
</tr>
<tr>
<td>TPM_RC_NV_RANGE</td>
<td>if TPMA_NV_WRITEALL is SET then the write is not the size of the Index referenced by nvIndex; otherwise, the write extends beyond the limits of the Index</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_NV_Write(
    NV_Write_In *in // IN: input parameter list
)
{
    NV_INDEX nvIndex;
    TPM_RC result;

    // Input Validation

    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);

    // common access checks. NvWriteAccessChecks() may return
    // TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
    result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Bits index, extend index or counter index may not be updated by
    // TPM2_NV_Write
    if(
        nvIndex.publicArea.attributes.TPMA_NV_COUNTER == SET
        || nvIndex.publicArea.attributes.TPMA_NV_BITS == SET
        || nvIndex.publicArea.attributes.TPMA_NV_EXTEND == SET)
        return TPM_RC_ATTRIBUTES;

    // Too much data
    if((in->data.t.size + in->offset) > nvIndex.publicArea.dataSize)
        return TPM_RC_NV_RANGE;

    // If this index requires a full sized write, make sure that input range is
    // full sized
    if(
        nvIndex.publicArea.attributes.TPMA_NV_WRITEALL == SET
        && in->data.t.size < nvIndex.publicArea.dataSize)
        return TPM_RC_NV_RANGE;

    // Internal Data Update

    // Perform the write. This called routine will SET the TPMA_NV_WRITTEN
    // attribute if it has not already been SET. If NV isn't available, an error
    // will be returned.
    return NvWriteIndexData(in->nvIndex, &nvIndex, in->offset,
```
in->data.t.size, in->data.t.buffer);

}
33.8  TPM2_NV_Increment

33.8.1  General Description

This command is used to increment the value in an NV Index that has TPMA_NV_COUNTER SET. The data value of the NV Index is incremented by one.

NOTE 1  The NV Index counter is an unsigned value.

If TPMA_NV_COUNTER is not SET in the indicated NV Index, the TPM shall return TPM_RC_ATTRIBUTES.

If TPMA_NV_WRITELOCKED is SET, the TPM shall return TPM_RC_NV_LOCKED.

If TPMA_NV_WRITTEN is CLEAR, it will be SET.

If TPMA_NV_ORDERLY is SET, and the difference between the volatile and non-volatile versions of this field is greater than MAX_ORDERLY_COUNT, then the non-volatile version of the counter is updated.

NOTE 2  If a TPM implements TPMA_NV_ORDERLY and an Index is defined with TPMA_NV_ORDERLY and TPM_NV_COUNTER both SET, then in the Event of a non-orderly shutdown, the non-volatile value for the counter Index will be advanced by MAX_ORDERLY_COUNT at the next TPM2_Startup().

NOTE 3  An allowed implementation would keep a counter value in NV and a resettable counter in RAM. The reported value of the NV Index would be the sum of the two values. When the RAM count increments past the maximum allowed value (MAX_ORDERLY_COUNT), the non-volatile version of the count is updated with the sum of the values and the RAM count is reset to zero.
33.8.2 Command and Response

Table 207 — TPM2_NV_Increment Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_Increment {NV} handle indicating the source of the authorization value</td>
</tr>
<tr>
<td>TPMI_RH_NV_AUTH</td>
<td>@authHandle</td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>the NV Index to increment Auth Index: None</td>
</tr>
</tbody>
</table>

Table 208 — TPM2_NV_Increment Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.8.3 Detailed Actions

1. #include "InternalRoutines.h"
2. #include "NV_Increment_fp.h"
3. #include "NV_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>NV index is not a counter</td>
</tr>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>authorization failure</td>
</tr>
<tr>
<td>TPM_RC_NV_LOCKED</td>
<td>Index is write locked</td>
</tr>
</tbody>
</table>

4. TPM_RC
5. TPM2_NV_Increment(
6.   NV_Increment_In *in        // IN: input parameter list
7. )
8. {
9.   TPM_RC                  result;
10.  NV_INDEX               nvIndex;
11.  UINT64                  countValue;
12. // Input Validation
13. // Common access checks, a TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
14. // error may be returned at this point
15. result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
16. if(result != TPM_RC_SUCCESS)
17.   return result;
18. // Get NV index info
19. NvGetIndexInfo(in->nvIndex, &nvIndex);
20. // Make sure that this is a counter
21. if(nvIndex.publicArea.attributes.TPMA_NV_COUNTER != SET)
22.   return TPM_RC_ATTRIBUTES + RC_NV_Increment_nvIndex;
23. // Internal Data Update
24. // If counter index is not been written, initialize it
25. if(nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == CLEAR)
26.   countValue = NvInitialCounter();
27. else
28.   // Read NV data in native format for TPM CPU.
29.   NvGetIntIndexData(in->nvIndex, &nvIndex, &countValue);
30. // Do the increment
31.   countValue++;
32. // If this is an orderly counter that just rolled over, need to be able to
33. // write to NV to proceed. This check is done here, because NvWriteIndexData()
34. // does not see if the update is for counter rollover.
35. if( nvIndex.publicArea.attributes.TPMA_NV_ORDERLY == SET
36.     & (countValue & MAX_ORDERLY_COUNT) == 0)
37.   { result = NvIsAvailable();
38.     if(result != TPM_RC_SUCCESS)
39.       return result;
40.   } // Need to force an NV update
52     g_updateNV = TRUE;
53 }
54
55 // Write NV data back. A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may
56 // be returned at this point. If necessary, this function will set the
57 // TPMA_NV_WRITTEN attribute
58 return NvWriteIndexData(in->nvIndex, &nvIndex, 0, 8, &countValue);
33.9 TPM2_NV_Extend

33.9.1 General Description

This command extends a value to an area in NV memory that was previously defined by TPM2_NV_DefineSpace.

If TPMA_NV_EXTEND is not SET, then the TPM shall return TPM_RC_ATTRIBUTES.

Proper write authorizations are required for this command as determined by TPMA_NV_PPWRITE, TPMA_NV_OWNERWRITE, TPMA_NV_AUTHWRITE, and the authPolicy of the NV Index.

After successful completion of this command, TPMA_NV_WRITTEN for the NV Index will be SET.

NOTE 1 Once SET, TPMA_NV_WRITTEN remains SET until the NV Index is undefined or the NV Index is cleared.

If the TPMA_NV_WRITELOCKED attribute of the NV Index is SET, then the TPM shall return TPM_RC_NV_LOCKED.

NOTE 2 If authorization sessions are present, they are checked before checks to see if writes to the NV Index are locked.

The data.buffer parameter may be larger than the defined size of the NV Index.

The Index will be updated by:

\[ \text{nvIndex} \rightarrow \text{data}_{\text{new}} := H_{\text{nameAlg}}(\text{nvIndex} \rightarrow \text{data}_{\text{old}} \mid \text{data.buffer}) \]  

(39)

where

- \( H_{\text{nameAlg}}() \) the hash algorithm indicated in \( \text{nvIndex} \rightarrow \text{nameAlg} \)
- \( \text{nvIndex} \rightarrow \text{data} \) the value of the data field in the NV Index
- \( \text{data.buffer} \) the data buffer of the command parameter

NOTE 3 If TPMA_NV_WRITTEN is CLEAR, then \( \text{nvIndex} \rightarrow \text{data} \) is a Zero Digest.
### 33.9.2 Command and Response

#### Table 209 — TPM2_NV_Extend Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_Extend (NV)</td>
</tr>
<tr>
<td>TPMI_RH_NV_AUTH</td>
<td>@authHandle</td>
<td>handle indicating the source of the authorization value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPML_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>the NV Index to extend</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
<tr>
<td>TPM2B_MAX_NV_BUFFER</td>
<td>data</td>
<td>the data to extend</td>
</tr>
</tbody>
</table>

#### Table 210 — TPM2_NV_Extend Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
### 33.9.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_Extend_fp.h"
#include "NV_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>the TPMA_NV_EXTEND attribute is not SET in the Index referenced by nvIndex</td>
</tr>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>the authorization was valid but the authorizing entity (authHandle) is not allowed to write to the Index referenced by nvIndex</td>
</tr>
<tr>
<td>TPM_RC_NV_LOCKED</td>
<td>the Index referenced by nvIndex is locked for writing</td>
</tr>
</tbody>
</table>

```c
TPM_RC
TPM2_NV_Extend(
    NV_Extend_In *in   // IN: input parameter list
) {
    TPM_RC result;
    NV_INDEX nvIndex;
    TPM2B_DIGEST oldDigest;
    TPM2B_DIGEST newDigest;
    HASH_STATE hashState;

    // Input Validation
    // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
    // or TPM_RC_NV_LOCKED
    result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);

    // Make sure that this is an extend index
    if(nvIndex.publicArea.attributes.TPMA_NV_EXTEND != SET)
        return TPM_RC_ATTRIBUTES + RC_NV_Extend_nvIndex;

    // If the Index is not-orderly, or if this is the first write, NV will
    // need to be updated.
    if( ! (nvIndex.publicArea.attributes.TPMA_NV_ORDERLY == CLEAR
        || nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == CLEAR) )
        {
            // Check if NV is available. NvIsAvailable may return TPM_RC_NV_UNAVAILABLE
            // TPM_RC_NV_RATE or TPM_RC_SUCCESS.
            result = NvIsAvailable();
            if(result != TPM_RC_SUCCESS)
                return result;
        }

    // Internal Data Update
    // Perform the write.
    oldDigest.t.size = CryptGetHashDigestSize(nvIndex.publicArea.nameAlg);
    if(nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == SET)
        {
            NvGetIndexData(in->nvIndex, &nvIndex, 0,
```
oldDigest.t.size, oldDigest.t.buffer);

} else {
    MemorySet(oldDigest.t.buffer, 0, oldDigest.t.size);
}

// Start hash
newDigest.t.size = CryptStartHash(nvIndex.publicArea.nameAlg, &hashState);

// Adding old digest
CryptUpdateDigest2B(&hashState, &oldDigest.b);

// Adding new data
CryptUpdateDigest2B(&hashState, &in->data.b);

// Complete hash
CryptCompleteHash2B(&hashState, &newDigest.b);

// Write extended hash back.
// Note, this routine will SET the TPMA_NV_WRITTEN attribute if necessary
return NvWriteIndexData(in->nvIndex, &nvIndex, 0,
    newDigest.t.size, newDigest.t.buffer);
33.10  TPM2_NV_SetBits

33.10.1  General Description

This command is used to SET bits in an NV Index that was created as a bit field. Any number of bits from 0 to 64 may be SET. The contents of data are ORed with the current contents of the NV Index starting at offset. The checks on data and offset are the same as for TPM2_NV_Write.

If TPMA_NV_WRITTEN is not SET, then, for the purposes of this command, the NV Index is considered to contain all zero bits and data is OR with that value.

If TPMA_NV_BITS is not SET, then the TPM shall return TPM_RC_ATTRIBUTES.

After successful completion of this command, TPMA_NV_WRITTEN for the NV Index will be SET.

NOTE  TPMA_NV_WRITTEN will be SET even if no bits were SET.
### 33.10.2 Command and Response

#### Table 211 — TPM2_NV_SetBits Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td>TPMI_ST_COMMAND_TAG</td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_SetBits (NV)</td>
</tr>
<tr>
<td>TPMI_RH_NV_AUTH</td>
<td>@authHandle</td>
<td>handle indicating the source of the authorization value</td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>NV Index of the area in which the bit is to be set</td>
</tr>
<tr>
<td>UINT64</td>
<td>bits</td>
<td>the data to OR with the current contents</td>
</tr>
</tbody>
</table>

#### Table 212 — TPM2_NV_SetBits Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.10.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_SetBits_fp.h"
#include "NV_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>the TPMA_NV_BITS attribute is not SET in the Index referenced by nvIndex</td>
</tr>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>the authorization was valid but the authorizing entity (authHandle)</td>
</tr>
<tr>
<td></td>
<td>is not allowed to write to the Index referenced by nvIndex</td>
</tr>
<tr>
<td>TPM_RC_NV_LOCKED</td>
<td>the Index referenced by nvIndex is locked for writing</td>
</tr>
</tbody>
</table>
```

```
TPM_RC
TPM2_NV_SetBits(
    NV_SetBits_In *in          // IN: input parameter list
)
{
    TPM_RC         result;
    NV_INDEX       nvIndex;
    UINT64         bitValue;

    // Input Validation
    // Common access checks, NvWriteAccessCheck() may return TPM_RC_NV_AUTHORIZATION
    // or TPM_RC_NV_LOCKED
    // error may be returned at this point
    result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);

    // Make sure that this is a bit field
    if(nvIndex.publicArea.attributes.TPMA_NV_BITS != SET)
        return TPM_RC_ATTRIBUTES + RC_NV_SetBits_nvIndex;

    // If the Index is not-orderly, or if this is the first write, NV will
    // need to be updated.
    if(nvIndex.publicArea.attributes.TPMA_NV_ORDERLY == CLEAR
        || nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == CLEAR)
    {
        // Check if NV is available. NvIsAvailable may return TPM_RC_NV_UNAVAILABLE
        // TPM_RC_NV_RATE or TPM_RC_SUCCESS.
        result = NvIsAvailable();
        if(result != TPM_RC_SUCCESS)
            return result;
    }

    // Internal Data Update
    // If index is not been written, initialize it
    if(nvIndex.publicArea.attributes.TPMA_NV_WRITTEN == CLEAR)
    {
        bitValue = 0;
    }

    else
        // Read index data
```

```
```
NvGetIntIndexData(in->nvIndex, &nvIndex, &bitValue);

// OR in the new bit setting
bitValue |= in->bits;

// Write index data back. If necessary, this function will SET
// TPMA_NV_WRITTEN.
return NvWriteIndexData(in->nvIndex, &nvIndex, 0, 8, &bitValue);
33.11 TPM2_NV_WriteLock

33.11.1 General Description

If the TPMA_NV_WRITEDEFINE or TPMA_NV_WRITE_STCLEAR attributes of an NV location are SET, then this command may be used to inhibit further writes of the NV Index.

Proper write authorization is required for this command as determined by TPMA_NV_PPWRITE, TPMA_NV_OWNERWRITE, TPMA_NV_AUTHWRITE, and the authPolicy of the NV Index.

It is not an error if TPMA_NV_WRITELOCKED for the NV Index is already SET.

If neither TPMA_NV_WRITEDEFINE nor TPMA_NV_WRITE_STCLEAR of the NV Index is SET, then the TPM shall return TPM_RC_ATTRIBUTES.

If the command is properly authorized and TPMA_NV_WRITE_STCLEAR or TPMA_NV_WRITEDEFINE is SET, then the TPM shall SET TPMA_NV_WRITELOCKED for the NV Index. TPMA_NV_WRITELOCKED will be clear on the next TPM2_Startup(TPM_SU_CLEAR) unless TPMA_NV_WRITEDEFINE is SET.
### 33.11.2 Command and Response

**Table 213 — TPM2_NV_WriteLock Command**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_WriteLock (NV) handle indicating the source of the authorization value</td>
</tr>
<tr>
<td>TPM_RH_NV_AUTH</td>
<td>@authHandle</td>
<td>Auth Index: 1, Auth Role: USER</td>
</tr>
<tr>
<td>TPM_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>the NV Index of the area to lock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
</tbody>
</table>

**Table 214 — TPM2_NV_WriteLock Response**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.11.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_WriteLock_fp.h"
#include "NV_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>neither TPMA_NV_WRITEDEFINE nor TPMA_NV_WRITE_STCLEAR is SET in Index referenced by nIndex</td>
</tr>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>the authorization was valid but the authorizing entity (authHandle) is not allowed to write to the Index referenced by nIndex</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_NV_WriteLock(
    NV_WriteLock_In *in          // IN: input parameter list
) {
    TPM_RC          result;
    NV_INDEX        nvIndex;

    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at this point.
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

    // Input Validation:
    // Common write access checks, a TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
    // error may be returned at this point
    result = NvWriteAccessChecks(in->authHandle, in->nvIndex);
    if(result != TPM_RC_SUCCESS)
        { if(result == TPM_RC_NV_AUTHORIZATION)
            return TPM_RC_NV_AUTHORIZATION;
            // If write access failed because the index is already locked, then it is no error.
            return TPM_RC_SUCCESS;
        }

    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);

    // if non of TPMA_NV_WRITEDEFINE or TPMA_NV_WRITE_STCLEAR is set, the index
    // can not be write-locked
    if(nvIndex.publicArea.attributes.TPMA_NV_WRITEDEFINE == CLEAR
        && nvIndex.publicArea.attributes.TPMA_NV_WRITE_STCLEAR == CLEAR)
        return TPM_RC_ATTRIBUTES + RC_NV_WriteLock_nvIndex;

    // Internal Data Update
    // Set the WRITELOCK attribute
    nvIndex.publicArea.attributes.TPMA_NV_WRITELOCKED = SET;

    // Write index info back
    NvWriteIndexInfo(in->nvIndex, &nvIndex);
```
51 return TPM_RC_SUCCESS;
52 }

33.12 TPM2_NV_GlobalWriteLock

33.12.1 General Description

The command will SET TPMA_NV_WRITELOCKED for all indexes that have their TPMA_NV_GLOBALLOCK attribute SET.

If an Index has both TPMA_NV_WRITELOCKED and TPMA_NV_WRITEDEFINE SET, then this command will permanently lock the NV Index for writing.

NOTE If an Index is defined with TPMA_NV_GLOBALLOCK SET, then the global lock does not apply until the next time this command is executed.

This command requires either platformAuth/platformPolicy or ownerAuth/ownerPolicy.
### 33.12.2 Command and Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_GlobalWriteLock</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TPML_RH_PROVISION @authHandle</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
</tbody>
</table>

#### Table 216 — TPM2_NV_GlobalWriteLock Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.12.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_GlobalWriteLock_fp.h"
TPM_RC
TPM2_NV_GlobalWriteLock(
    NV_GlobalWriteLock_In *in) // IN: input parameter list
{
    TPM_RC          result;
    // Input parameter is not reference in command action
    in = NULL; // to silence compiler warnings.
    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
    // this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS)
        return result;

    // Internal Data Update
    // Implementation dependent method of setting the global lock
    NvSetGlobalLock();
    return TPM_RC_SUCCESS;
}
```
33.13 TPM2_NV_Read

33.13.1 General Description

This command reads a value from an area in NV memory previously defined by TPM2_NV_DefineSpace().

Proper authorizations are required for this command as determined by TPMA_NV_PPREAD, TPMA_NV_OWNERREAD, TPMA_NV_AUTHREAD, and the authPolicy of the NV Index.

If TPMA_NV_READLOCKED of the NV Index is SET, then the TPM shall return TPM_RC_NV_LOCKED.

NOTE If authorization sessions are present, they are checked before the read-lock status of the NV Index is checked.

If the size parameter plus the offset parameter adds to a value that is greater than the size of the NV Index data area, the TPM shall return TPM_RC_NV_RANGE and not read any data from the NV Index.

If the NV Index has been defined but the TPMA_NV_WRITTEN attribute is CLEAR, then this command shall return TPM_RC_NV_UINITIALIZED even if size is zero.

The data parameter in the response may be encrypted using parameter encryption.
### 33.13.2 Command and Response

#### Table 217 — TPM2_NV_Read Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_Read</td>
</tr>
<tr>
<td>TPMI_RH_NV_AUTH</td>
<td>@authHandle</td>
<td>the handle indicating the source of the authorization value</td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>the NV Index to be read</td>
</tr>
<tr>
<td>UINT16</td>
<td>size</td>
<td>number of octets to read</td>
</tr>
<tr>
<td>UINT16</td>
<td>offset</td>
<td>octet offset into the area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value shall be less than or equal to the size of the nvIndex data.</td>
</tr>
</tbody>
</table>

#### Table 218 — TPM2_NV_Read Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_MAX_NV_BUFFER</td>
<td>data</td>
<td>the data read</td>
</tr>
</tbody>
</table>
33.13.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_Read_fp.h"
#include "NV_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>the authorization was valid but the authorizing entity (authHandle) is not allowed to read from the Index referenced by nvIndex</td>
</tr>
<tr>
<td>TPM_RC_NV_LOCKED</td>
<td>the Index referenced by nvIndex is read locked</td>
</tr>
<tr>
<td>TPM_RC_NV_RANGE</td>
<td>read range defined by size and offset is outside the range of the Index referenced by nvIndex</td>
</tr>
<tr>
<td>TPM_RC_NV_UNINITIALIZED</td>
<td>the Index referenced by nvIndex has not been initialized (written)</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_NV_Read(
    NV_Read_In *in,  // IN: input parameter list
    NV_Read_Out *out // OUT: output parameter list
)
{
    NV_INDEX nvIndex;
    TPM_RC result;

    // Input Validation
    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);
    // Common read access checks. NvReadAccessChecks() returns
    // TPM_RC_NV_AUTHORIZATION, TPM_RC_NV_LOCKED, or TPM_RC_NV_UNINITIALIZED
    // error may be returned at this point
    result = NvReadAccessChecks(in->authHandle, in->nvIndex);
    if(result != TPM_RC_SUCCESS)
        return result;

    // Too much data
    if((in->size + in->offset) > nvIndex.publicArea.dataSize)
        return TPM_RC_NV_RANGE;

    // Command Output
    // Set the return size
    out->data.t.size = in->size;
    // Perform the read
    NvGetIndexData(in->nvIndex, &nvIndex, in->offset, in->size, out->data.t.buffer);
    return TPM_RC_SUCCESS;
}
```
33.14 TPM2_NV_ReadLock

33.14.1 General Description

If TPMA_NV_READ_STCLEAR is SET in an Index, then this command may be used to prevent further reads of the NV Index until the next TPM2_Startup (TPM_SU_CLEAR).

Proper authorizations are required for this command as determined by TPMA_NV_PPREAD, TPMA_NV_OWNERREAD, TPMA_NV_AUTHREAD, and the authPolicy of the NV Index.

NOTE Only an entity that may read an Index is allowed to lock the NV Index for read.

If the command is properly authorized and TPMA_NV_READ_STCLEAR of the NV Index is SET, then the TPM shall SET TPMA_NV_READLOCKED for the NV Index. If TPMA_NV_READ_STCLEAR of the NV Index is CLEAR, then the TPM shall return TPM_RC_NV_ATTRIBUTE. TPMA_NV_READLOCKED will be CLEAR by the next TPM2_Startup(TPM_SU_CLEAR).

It is not an error to use this command for an Index that is already locked for reading.
An Index that had not been written may be locked for reading.
33.14.2 Command and Response

Table 219 — TPM2_NV_ReadLock Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPML_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_ReadLock</td>
</tr>
<tr>
<td>TPML_RH_NV_AUTH</td>
<td>@authHandle</td>
<td>the handle indicating the source of the authorization value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Role: USER</td>
</tr>
<tr>
<td>TPML_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>the NV Index to be locked</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Auth Index: None</td>
</tr>
</tbody>
</table>

Table 220 — TPM2_NV_ReadLock Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.14.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_ReadLock_fp.h"
#include "NV_spt_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_ATTRIBUTES</td>
<td>TPMA_NV_READ_STCLEAR is not SET so Index referenced by <code>nvIndex</code> may not be write locked</td>
</tr>
<tr>
<td>TPM_RC_NV_AUTHORIZATION</td>
<td>the authorization was valid but the authorizing entity (authHandle) is not allowed to read from the Index referenced by <code>nvIndex</code></td>
</tr>
</tbody>
</table>

TPM_RC
TPM2_NV_ReadLock(
    NV_ReadLock_In *in /* IN: input parameter list */
)
{
    TPM_RC result;
    NV_INDEX nvIndex;
    // The command needs NV update. Check if NV is available.
    // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at this point
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;
    // Input Validation
    // Common read access checks. NvReadAccessChecks() returns
    // TPM_RC_NV_AUTHORIZATION, TPM_RC_NV_LOCKED, or TPM_RC_NV_UNINITIALIZED
    // error may be returned at this point
    result = NvReadAccessChecks(in->authHandle, in->nvIndex);
    if(result != TPM_RC_SUCCESS)
    {
        if(result == TPM_RC_NV_AUTHORIZATION)
            return TPM_RC_NV_AUTHORIZATION;
        // Index is already locked for write
        else if(result == TPM_RC_NV_LOCKED)
            return TPM_RC_SUCCESS;
    }
    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);
    // If NvReadAccessChecks return TPM_RC_NV_UNINITIALIZED, then continue.
    // It is not an error to read lock an uninitialized Index.
    // Set the READLOCK attribute
    nvIndex.publicArea.attributes.TPMA_NV_READLOCKED = SET;
    // Write NV info back
    NvWriteIndexInfo(in->nvIndex, &nvIndex);
    return TPM_RC_SUCCESS;
}
```
33.15 TPM2_NV_ChangeAuth

33.15.1 General Description

This command allows the authorization secret for an NV Index to be changed. If successful, the authorization secret (authValue) of the NV Index associated with nvIndex is changed. This command requires that a policy session be used for authorization of nvIndex so that the ADMIN role may be asserted and that commandCode in the policy session context shall be TPM_CC_NV_ChangeAuth. That is, the policy must contain a specific authorization for changing the authorization value of the referenced object.

NOTE The reason for this restriction is to ensure that the administrative actions on nvIndex require explicit approval while other commands may use policy that is not command-dependent.

The size of the newAuth value may be no larger than the size of authorization indicated when the NV Index was defined.

Since the NV Index authorization is changed before the response HMAC is calculated, the newAuth value is used when generating the response HMAC key if required. See Part 4 ComputeResponseHMAC().
33.15.2 Command and Response

Table 221 — TPM2_NV_ChangeAuth Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_ChangeAuth (NV)</td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>@nvIndex</td>
<td>handle of the object</td>
</tr>
<tr>
<td>TPM2B_AUTH</td>
<td>newAuth</td>
<td>new authorization value</td>
</tr>
</tbody>
</table>

Table 222 — TPM2_NV_ChangeAuth Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
</tbody>
</table>
33.15.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "NV_ChangeAuth_fp.h"

<table>
<thead>
<tr>
<th>Error Returns</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_RC_SIZE</td>
<td>newAuth size is larger than the digest size of the Name algorithm for the Index referenced by 'nvIndex'</td>
</tr>
</tbody>
</table>

TPM_RC

TPM2_NV_ChangeAuth(
    NV_ChangeAuth_In *in        // IN: input parameter list
) {
    TPM_RC result;
    NV_INDEX nvIndex;

    // Input Validation
    // Check if NV is available. NvIsAvailable may return TPM_RC_NV_UNAVAILABLE
    // TPM_RC_NV_RATE or TPM_RC_SUCCESS.
    result = NvIsAvailable();
    if(result != TPM_RC_SUCCESS) return result;

    // Read index info from NV
    NvGetIndexInfo(in->nvIndex, &nvIndex);

    // Remove any trailing zeros that might have been added by the caller
    // to obfuscate the size.
    MemoryRemoveTrailingZeros(&(in->newAuth));

    // Make sure that the authValue is no larger than the nameAlg of the Index
    if(in->newAuth.t.size > CryptGetHashDigestSize(nvIndex.publicArea.nameAlg))
        return TPM_RC_SIZE + RC_NV_ChangeAuth_newAuth;

    // Internal Data Update
    // Change auth
    nvIndex.authValue = in->newAuth;
    // Write index info back to NV
    NvWriteIndexInfo(in->nvIndex, &nvIndex);

    return TPM_RC_SUCCESS;
}
```
33.16 TPM2_NV_Certify

33.16.1 General Description

The purpose of this command is to certify the contents of an NV Index or portion of an NV Index.

If proper authorization for reading the NV Index is provided, the portion of the NV Index selected by size and offset are included in an attestation block and signed using the key indicated by signHandle. The attestation also includes size and offset so that the range of the data can be determined.

NOTE See 20.1 for description of how the signing scheme is selected.
33.16.2 Command and Response

Table 223 — TPM2_NV_Certify Command

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPMI_ST_COMMAND_TAG</td>
<td>tag</td>
<td></td>
</tr>
<tr>
<td>UINT32</td>
<td>commandSize</td>
<td></td>
</tr>
<tr>
<td>TPM_CC</td>
<td>commandCode</td>
<td>TPM_CC_NV_Certify</td>
</tr>
<tr>
<td>TPMI_DH_OBJECT+</td>
<td>signHandle</td>
<td>handle of the key used to sign the attestation structure</td>
</tr>
<tr>
<td></td>
<td>Auth Index: 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Auth Role: USER</td>
<td></td>
</tr>
<tr>
<td>TPMI_RH_NV_AUTH</td>
<td>authHandle</td>
<td>handle indicating the source of the authorization value for the NV Index</td>
</tr>
<tr>
<td></td>
<td>Auth Index: 2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Auth Role: USER</td>
<td></td>
</tr>
<tr>
<td>TPMI_RH_NV_INDEX</td>
<td>nvIndex</td>
<td>Index for the area to be certified</td>
</tr>
<tr>
<td></td>
<td>Auth Index: None</td>
<td></td>
</tr>
<tr>
<td>TPM2B_DATA</td>
<td>qualifyingData</td>
<td>user-provided qualifying data</td>
</tr>
<tr>
<td>TPMT_SIG_SCHEME+</td>
<td>inScheme</td>
<td>signing scheme to use if the scheme for signHandle is TPM_ALG_NULL</td>
</tr>
<tr>
<td>UINT16</td>
<td>size</td>
<td>number of octets to certify</td>
</tr>
<tr>
<td>UINT16</td>
<td>offset</td>
<td>octet offset into the area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This value shall be less than or equal to the size of the nvIndex data.</td>
</tr>
</tbody>
</table>

Table 224 — TPM2_NV_Certify Response

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPM_ST</td>
<td>tag</td>
<td>see clause 8</td>
</tr>
<tr>
<td>UINT32</td>
<td>responseSize</td>
<td></td>
</tr>
<tr>
<td>TPM_RC</td>
<td>responseCode</td>
<td></td>
</tr>
<tr>
<td>TPM2B_ATTEST</td>
<td>certifyInfo</td>
<td>the structure that was signed</td>
</tr>
<tr>
<td>TPMT_SIGNATURE</td>
<td>signature</td>
<td>the asymmetric signature over certifyInfo using the key referenced by signHandle</td>
</tr>
</tbody>
</table>
33.16.3 Detailed Actions

```c
#include "InternalRoutines.h"
#include "Attest_spt_fp.h"
#include "NV_spt_fp.h"
#include "NV_Certify_fp.h"

Error Returns | Meaning
---|---
TPM_RC_NV_AUTHORIZATION | the authorization was valid but the authorizing entity (authHandle) is not allowed to read from the Index referenced by nvIndex
TPM_RC_KEY | signHandle does not reference a signing key
TPM_RC_NV_LOCKED | Index referenced by nvIndex is locked for reading
TPM_RC_NV_RANGE | offset plus size extends outside of the data range of the Index referenced by nvIndex
TPM_RC_NV_UNINITIALIZED | Index referenced by nvIndex has not been written
TPM_RC_SCHEME | inScheme is not an allowed value for the key definition

TPM_RC
TPM2_NV_Certify(
    NV_Certify_In    *in,       // IN: input parameter list
    NV_Certify_Out   *out        // OUT: output parameter list
)
{
    TPM_RC                  result;
    NV_INDEX                nvIndex;
    TPMS_ATTEST             certifyInfo;

    // Attestation command may cause the orderlyState to be cleared due to
    // the reporting of clock info.  If this is the case, check if NV is
    // available first
    if(gp.orderlyState != SHUTDOWN_NONE)
    {
        // The command needs NV update.  Check if NV is available.
        // A TPM_RC_NV_UNAVAILABLE or TPM_RC_NV_RATE error may be returned at
        // this point
        result = NvIsAvailable();
        if(result != TPM_RC_SUCCESS)
            return result;
    }

    // Input Validation
    // Get NV index info
    NvGetIndexInfo(in->nvIndex, &nvIndex);
    // Common access checks.  A TPM_RC_NV_AUTHORIZATION or TPM_RC_NV_LOCKED
    // error may be returned at this point
    result = NvReadAccessChecks(in->authHandle, in->nvIndex);
    if(result != TPM_RC_SUCCESS)
        return result;

    // See if the range to be certified is out of the bounds of the defined
    // Index
    if((in->size + in->offset) > nvIndex.publicArea.dataSize)
        return TPM_RC_NV_RANGE;

    // Command Output
```
// Filling in attest information
// Common fields
// FillInAttestInfo can return TPM_RC_SCHEME or TPM_RC_KEY
result = FillInAttestInfo(in->signHandle,
   &in->inScheme,
   &in->qualifyingData,
   &certifyInfo);
if(result != TPM_RC_SUCCESS) {
    if(result == TPM_RC_KEY)
        return TPM_RC_KEY + RC_NV_Certify_signHandle;
    else
        return RcSafeAddToResult(result, RC_NV_Certify_inScheme);
}
// NV certify specific fields
// Attestation type
certifyInfo.type = TPM_ST_ATTEST_NV;

// Get the name of the index
certifyInfo.attested.nv.indexName.t.size =
   NvGetName(in->nvIndex, &certifyInfo.attested.nv.indexName.t.name);

// Set the return size
certifyInfo.attested.nv.nvContents.t.size = in->size;

// Set the offset
certifyInfo.attested.nv.offset = in->offset;

// Perform the read
NvGetIndexData(in->nvIndex, &nvIndex,
   in->offset, in->size,
   certifyInfo.attested.nv.nvContents.t.buffer);

// Sign attestation structure. A NULL signature will be returned if
// signHandle is TPM_RH_NULL. SignAttestInfo() may return TPM_RC_VALUE,
// TPM_RC_SCHEME or TPM_RC_ATTRIBUTES.
// Note: SignAttestInfo may return TPM_RC_ATTRIBUTES if the key is not a
// signing key but that was checked above. TPM_RC_VALUE would mean that the
// data to sign is too large but the data to sign is a digest
result = SignAttestInfo(in->signHandle,
   &in->inScheme,
   &certifyInfo,
   &in->qualifyingData,
   &out->certifyInfo,
   &out->signature);
if(result != TPM_RC_SUCCESS) return result;

// orderly state should be cleared because of the reporting of clock info
// if signing happens
if(in->signHandle != TPM_RH_NULL)
    g_clearOrderly = TRUE;
return TPM_RC_SUCCESS;