TCG PC Client Specific TPM Interface Specification (TIS)

Version 1.2 FINAL
Revision 1.00
July 11, 2005
For TPM Family 1.2; Level 2

Change History

Revision	Date	Description
1.00	July 11, 2005	Initial release.

Contents

Figures

Tables

Corrections and Comments

TCG members may send comments to:
techquestions@trustedcomputinggroup.org

TPM Dependency and Requirements

The TPM used for Host Platforms claiming adherence to this specification MUST be compliant with the *TPM Main Specification; Family 1.2; Level 2; Revision 0.85* or later.

1. TPM Requirements General Introduction

Start of informative comment

The TCG architecture specifications define a TPM for use on any non-platform specific platform. However, due to the requirements to support Dynamic Locality features specific to the PC Client are necessary for the TPM. These PC Client features are defined in the section.

End of informative comment

The following are the additional features that MUST be implemented by a TPM for a PC Client as defined in this specification.

Unless otherwise indicated, the features in this specification are based on the *TPM Main Specification Family 1.2; Level 2; Revision 0.85* parts 1 through 3. The term TPM Main Specification SHALL reference these documents and the features they specify.

1.1 Terminology

The term “software” will refer generically to the entity sending commands to the TPM. The actual source of the command may be hardware, firmware, an operating system driver, or an application.

1.2 Division of Documentation

Start of informative comment

The PC Client Specifications are divided into two documents:

1. This one, the *PC Client Interface Specification*, which discusses the specifics regarding the requirements of the TPM for the PC Client but only to the requirements for the TPM itself. This document discusses the details of what interfaces and protocols are used to communicate with the TPM and the platform-specific set of requirements. Items such as the minimum number of PCRs required and NV Storage available are discussed. The target audience for this document is the TPM manufacturers but platform manufacturers should review it as well.
2. The *PC Client Implementation Specification* specifies the requirements for the TPM as it is implemented on the platform. Issues such as PCR mapping, functional interfaces, pre-operating system driver functionality, and interfaces are discussed.

End of informative comment

2. Summary of TPM Features to Support the PC Client

2.1 Locality

"Locality" is a concept that allows various trusted processes on the platform to communicate with the TPM such that the TPM is aware of which trusted process is sending commands. This is implemented using dedicated ranges of LPC addresses and a new LPC "start" field. There are six Localities defined (numbers 0 – 4 and Legacy). Their use is defined as:

1. Locality 4: Trusted Hardware. This is the Dynamic RTM.[1]
2. Locality 3: Auxiliary components. Use of this is optional and, if used, it is implementation dependent.
3. Locality 2: This is the "runtime" environment for the Trusted Operating System.
4. Locality 1: An environment for use by the Trusted Operating System (T/OS).
5. Locality 0: The legacy environment for the Static RTM and its chain of trust.
6. Locality Legacy: This is Locality 0 using TPM 1.1 type I/O ports.

See Section 9 for more details.

2.2 Resetable PCRs

Resetable PCRs is a set of PCRs for use by the Dynamic RTM and its chain of trust. Access control to these PCRs is by use of the various locality indicators.

2.3 Minimum Amount of NV Storage is Specified

The *TPM Main Specification* provides for a general-purpose area of Non-volatile storage for use by the platforms. The definition of this area is the purview of the various platform specific specifications. This specification will define the use for the PC Client.

2.4 Minimum Number of PCRs

The *TPM Main Specification* allows the platform specific specifications to require a minimum number of PCRs and to allocate usage for them based on the needs and the environment of the platform.

[1] Reference the *PC Client Implementation Specification* for the definition of Dynamic RTM.

3. Ordinal Table

Start of informative comment

The *TCG Main Specification* defines all functions needed for all types of platforms in a platform non-specific manner. Some of these defined functions are either not applicable or appropriate for some types of platforms, and it is left to the platform specific specifications to enumerate which of the TPM commands are to be required, optional, or prohibited for that type of platform.

End of informative comment

To be conformant to this specification, the TPM MUST adhere to the following table:.

Table 1: Ordinal Table for TPM Commands

Function (by Ordinal Identifier)	**M = Mandatory** **O = Optional** **X = Deleted in TPM 1.2**
TPM_ORD_ActivateIdentity	M
TPM_ORD_AuthorizeMigrationKey	M
TPM_ORD_CertifyKey	M
TPM_ORD_CertifyKey2	M
TPM_ORD_CertifySelfTest	X
TPM_ORD_ChangeAuth	M
TPM_ORD_ChangeAuthAsymFinish	O
TPM_ORD_ChangeAuthAsymStart	O
TPM_ORD_ChangeAuthOwner	M
TPM_ORD_CMK_ApproveMA	M
TPM_ORD_CMK_ConvertMigration	M
TPM_ORD_CMK_CreateBlob	M
TPM_ORD_CMK_CreateKey	M
TPM_ORD_CMK_CreateTicket	M
TPM_ORD_CMK_SetRestrictions	M
TPM_ORD_ContinueSelfTest	M
TPM_ORD_ConvertMigrationBlob	M
TPM_ORD_CreateCounter	M
TPM_ORD_CreateEndorsementKeyPair	M
TPM_ORD_CreateMaintenanceArchive	O1
TPM_ORD_CreateMigrationBlob	M
TPM_ORD_CreateRevocableEK	O
TPM_ORD_CreateWrapKey	M
TPM_ORD_DAA_JOIN	M
TPM_ORD_DAA_SIGN	M
TPM_ORD_Delegate_CreateKeyDelegation	M

Function (by Ordinal Identifier)	**M = Mandatory** **O = Optional** **X = Deleted in TPM 1.2**
TPM_ORD_Delegate_CreateOwnerDelegation	M
TPM_ORD_Delegate_LoadOwnerDelegation	M
TPM_ORD_Delegate_Manage	M
TPM_ORD_Delegate_ReadTable	M
TPM_ORD_Delegate_UpdateVerification	M
TPM_ORD_Delegate_VerifyDelegation	M
TPM_ORD_DirRead	O
TPM_ORD_DirWriteAuth	O
TPM_ORD_DisableForceClear	M
TPM_ORD_DisableOwnerClear	M
TPM_ORD_DisablePubekRead	O
TPM_ORD_DSAP	M
TPM_ORD_EstablishTransport	M
TPM_ORD_EvictKey	O
TPM_ORD_ExecuteTransport	M
TPM_ORD_Extend	M
TPM_ORD_FieldUpgrade	O
TPM_ORD_FlushSpecific	M
TPM_ORD_ForceClear	M
TPM_ORD_GetAuditDigest	O2
TPM_ORD_GetAuditDigestSigned	O2
TPM_ORD_GetAuditEvent	X
TPM_ORD_GetAuditEventSigned	X
TPM_ORD_GetCapability	M
TPM_ORD_GetCapabilityOwner	M
TPM_ORD_GetCapabilitySigned	X
TPM_ORD_GetOrdinalAuditStatus	X
TPM_ORD_GetPubKey	M
TPM_ORD_GetRandom	M
TPM_ORD_GetTestResult	M
TPM_ORD_GetTick	M
TPM_ORD_IncrementCounter	M
TPM_ORD_Init	M
TPM_ORD_KeyControlOwner	M
TPM_ORD_KillMaintenanceFeature	O1
TPM_ORD_LoadAuthContext	O
TPM_ORD_LoadContext	M
TPM_ORD_LoadKey	O

Function (by Ordinal Identifier)	M = Mandatory
	O = Optional
	X = Deleted in TPM 1.2

TPM_ORD_LoadKey2	M
TPM_ORD_LoadKeyContext	O
TPM_ORD_LoadMaintenanceArchive	O1
TPM_ORD_LoadManuMaintPub	O1
TPM_ORD_MakeIdentity	M
TPM_ORD_MigrateKey	M
TPM_ORD_NV_DefineSpace	M
TPM_ORD_NV_ReadValue	M
TPM_ORD_NV_ReadValueAuth	M
TPM_ORD_NV_WriteValue	M
TPM_ORD_NV_WriteValueAuth	M
TPM_ORD_OIAP	M
TPM_ORD_OSAP	M
TPM_ORD_OwnerClear	M
TPM_ORD_OwnerReadInternalPub	M
TPM_ORD_OwnerReadPubek	O
TPM_ORD_OwnerSetDisable	M
TPM_ORD_PCR_Reset	M
TPM_ORD_PcrRead	M
TPM_ORD_PhysicalDisable	M
TPM_ORD_PhysicalEnable	M
TPM_ORD_PhysicalSetDeactivated	M
TPM_ORD_Quote	M
TPM_ORD_Quote2	M
TPM_ORD_ReadCounter	M
TPM_ORD_ReadManuMaintPub	O1
TPM_ORD_ReadPubek	M
TPM_ORD_ReleaseCounter	M
TPM_ORD_ReleaseCounterOwner	M
TPM_ORD_ReleaseTransportSigned	M
TPM_ORD_Reset	O
TPM_ORD_ResetLockValue	M
TPM_ORD_RevokeTrust	O
TPM_ORD_SaveAuthContext	O
TPM_ORD_SaveContext	M
TPM_ORD_SaveKeyContext	O
TPM_ORD_SaveState	M
TPM_ORD_Seal	M

Function (by Ordinal Identifier)	**M = Mandatory** **O = Optional** **X = Deleted in TPM 1.2**
TPM_ORD_Sealx	O
TPM_ORD_SelfTestFull	M
TPM_ORD_SetCapability	M
TPM_ORD_SetOperatorAuth	M
TPM_ORD_SetOrdinalAuditStatus	O
TPM_ORD_SetOwnerInstall	M
TPM_ORD_SetOwnerPointer	M
TPM_ORD_SetRedirection	O
TPM_ORD_SetTempDeactivated	M
TPM_ORD_SHA1Complete	M
TPM_ORD_SHA1CompleteExtend	M
TPM_ORD_SHA1Start	M
TPM_ORD_SHA1Update	M
TPM_ORD_Sign	M
TPM_ORD_Startup	M
TPM_ORD_StirRandom	M
TPM_ORD_TakeOwnership	M
TPM_ORD_Terminate_Handle	O
TPM_ORD_TickStampBlob	M
TPM_ORD_UnBind	M
TPM_ORD_Unseal	M

Table Note:

O1: For each of these flags, if any of these functions are implemented, all those containing the same flag MUST become mandatory.

70 The connection commands manage the TPM's connection to the Trusted Building Block (TBB). See the *PC Client Implementation Specification* for a description of the TBB.

Table 2: Ordinal Table for TPM Connection Commands

Function (by Ordinal Identifier)	**Flagged as Optional in the *TCG Main Specification***	**M = Mandatory** **O = Optional**
TSC_ORD_PhysicalPresence		M
TSC_ORD_ResetEstablishmentBit		M

4. Power Management

Start of informative comment

While allowed, there is no requirement or provision for the TPM to enter ACPI device power management states other than D0 and D3. This was done to simplify the TPM's interactions with the platform's components including the software. The TPM_SaveState and TPM_Startup commands were created as a mechanism for the platform's components to communicate entry into and exit from the D3 Power State. The TPM_SaveState command allows even untrusted software to indicate to the TPM that the platform may enter a low power state where the TPM may enter into the D3 power state. The use of the term may is significant in that there is no requirement for the platform to actually enter the low power state after sending the TPM_SaveState command. The software may, in fact, send subsequent commands after sending the TPM_SaveState commands. The TPM_SaveState command simply informs the TPM to save the required volatile contents because power to the TPM may be removed at any time. The TPM is responsible for tracking its internal states so that if a command that alters the TPM's saved state is sent to the TPM after a TPM_SaveState command, the TPM must void the saved internal state so a subsequent TPM_Startup(ST_STATE) will fail.

It is the responsibility of the S-CRTM to indicate to the TPM using the TPM_Startup command whether the TPM must reset or restore its saved state (e.g., PCR values, etc.). If the S-CRTM indicates to restore the saved state (i.e., ST_STATE), this effectively restores the transitive trust chain. If the S-CRTM indicates to reset the saved state (i.e., ST_STATE), this effectively clears and restarts a new transitive trust state. The rationale here is the S-CRTM is trusted to establish the initial transitive trust chain so it should also be trusted to determine whether to restore or clear it.

End of informative comment

1. After TPM_Startup, the TPM MUST behave as if it is in ACPI Device Power State D0 even if it supports ACPI Device Power States D1-D2.
2. Upon exit from the ACPI D3 Power State, the TPM MUST fail all commands except TPM_Init.

5. Non-volatile Storage

Start of informative comment

The Non-volatile (NV) Storage provides a general-purpose data storage area for persistent data. The TPM provides the ability to add access control to this area for security or privacy reasons. This area is addressed using indexes.

While this area provides a general-purpose storage area for interoperability, some index values are predefined. There is, however, no requirement to use or write to the space addressed by these indexes.

Because this data is opaque and not interpreted by the TPM, there is no enforcement of its contents. However, this PC Client Specification requires these indexes, if used, to contain the indicated values.

NV Storage is also used to access the TPM's General Purpose I/O. For details on this usage, see Section 6. This usage allocates indices, not storage, so it consumes no actual data storage; therefore, this does not add to the NV Storage Size described in Section 5.1.

End of informative comment

5.1 NV Storage Size

Start of informative comment

Providing an adequate minimum amount of storage space is difficult to predict based on future and unspecified use of the platform. However, it is prudent to provide for some minimum and predictable amount of storage to allow processes to budget their allocation.

End of informative comment

A conformant TPM for the PC Client SHALL provide a minimum of 1280 bytes of NV Storage.

6. General Purpose I/O (GPIO)

Start of informative comment

General purpose I/O (GPIO) provides an interface between the TPM's command interface and an external device. The actual use and protocol of the signal is neither determined nor specified by this specification. Only a single GPIO pin is currently defined and it is optional.

The TPM's command interface accesses the GPIO pins using the NV Storage interface. This is much like a "memory-mapped" I/O in other architectures. The *TPM Main Specification* reserved 256 indices for this purpose tagged TPM_NV_INDEX_GPIO_xx where xx is the range 00-FF. Each index can be associated with a GPIO pin per the platform-specific specifications and to their final device destination. The *PC Client TPM Interface Specification* will only specify the routing of the GPIO index to the GPIO pin. It is the purview of the *PC Client Implementation Specifications* to specify the routing to the specific device. In general, this is done by mapping the data sent in the TPM_NV_WriteValue* command's data field to the associated GPIO pin(s).

Because GPIO can be used for security or privacy functions, it must not be open, by default, for public access. For this reason, it is required that the NV Storage area that is mapped to the GPIO be "defined" (we are using the term "defined" in the TCG's technical context of NV Storage) like any other NV Storage area prior to allowing its use. When defining this area, the TPM Owner may elect to assign rights per the normal TPM_NV_ATTRIBUTES definitions. Note that this is done only once until the TPM Owner is removed; at which time, the area returns to undefined and must again be defined again before use. The reason for this is that the new TPM Owner may have different security and privacy requirements for this GPIO.

The range reserved for GPIO is not specific to a particular platform. It is, therefore, a requirement that software or other platform processes using GPIOs understand the nature of the platform before using it (i.e., which NV Storage Index is associated with which GPIO and the purpose of the GPIO on that particular platform).

Note to Implementers

Careful examination of the TPM_NV_ATTRIBUTES reveals that if none of the reads or writes permission bits (1-2 and 16-18) are set, this area is set to public reads and writes. Also, note the use of negative logic as stated above. Therefore, a write of "1" to the GPIO-Express-00 bit deactivates the TCS while a write of a "0" to the GPIO-Express-00 bit activates the TCS.

(Editor's Note: The above is copy of text in the relevant section in the PC Client Implementation Specification for the convenience of the reader. Any changes to the above text should be reflected in that specification as well.)

Note that the pin-out specified in Section 14 is only recommended and is not mandatory. TPMs are allowed to be implemented using any packaging. However, if this packaging is chosen, the pins, including the location of the GPIO pins, are mandatory. If this packaging is not used, the TPM manufacturer must provide documentation to the platform manufacturer indicating which pin is used as these GPIO pins. For ease in documentation, regardless of whether the TPM implements the recommended packages or uses their own, the designation of this pin will be GPIO-Express-00.

Electrical Characteristics

The Trusted Configuration Space is defined as being active or enabled when the TCS_EN pin is *high*, and the Trusted Configuration Space is defined as being inactive or disabled when the TCS_EN pin is *low*. The value of *low* and *high* are defined in the *PCI Express Card Electromechanical Specification* and are also referenced in the *Low Pin Count (LPC) Interface Specification* which forms the basis for this specification's bus characteristics.

End of informative comment

Implementation of this section is optional; but if implemented, it is mandatory that it be done in the manner specified in this section.

6.1 Reserved NV Storage Indices for GPIO

Start of informative comment

To allow for both standardized and innovative uses of the GPIO feature, the indices are divided into two areas: Defined and Vendor Specified. Within the Defined area, only the specific use is allowed. Indices within this range not currently defined and tagged as reserved must not be used. Within the Vendor Specified set of indices, Vendors (either TPM or platform manufacturers) are allowed to use these for any purpose. It must be understood that multiple vendors may choose to use the same NV Storage index (or set of indices) for different purposes; therefore, any software that uses this area will be TPM or platform manufacturer specific.

End of informative comment

The TPM and platform manufacturers MUST use Table 3 when allocating NV Storage Indices for TPM_NV_INDEX_GPIO_xx.

Table 3: Reserved NV Storage Indices for GPIO

TPM_NV_INDEX_GPIO_xx where xx is:	Destination or Use	Description
00	GPIO-Express-00	MUST only be used as defined in Section 6.1.
01 – 7F	Reserved	MUST NOT be used
80 – FF	Vendor Specified	MAY be used for vendor-specific purposes. Use is not standardized.

6.2 NV Storage Index to GPIO Mapping

Start of informative comment

Currently the only defined usage of the GPIO is for use by the GPIO-Express-00 pin which allows software to control enabling of a feature of PCI Express using the TCS_EN pin of the PCI Express Root Complex per the *PCI Express Trusted Configuration Space ECR*. This enabling is not always required by the platform's specific architecture and design; but if this signal is required, it must be implemented as described in this section.

[Editor's Note: The above is a copy of texts from the relevant sections in the PC Client Interface Specification for the convenience of the reader. Any changes to the above text should be reflected in that specification as well.]

Setting of this pin to a low enables security and privacy features; therefore, it must always default to high and must remain high until explicitly set low using a TPM_NV_WriteValue*.

Special consideration of this behavior is necessary for TPMs that implement non-D0 ACPI device states to be certain that the value GPIO-Express-00 pin does not float to a high value caused only by the transition to a lower power state.

The *PC Client Implementation Specifications* describe the optional connection of this pin to platform components. TPM manufacturers implementing this feature, especially those not implementing the standard TPM pinout, should review the relevant sections in the *PC Client Implementation Specification*.

End of informative comment

1. The TPM MAY implement the GPIO functionality specified in this section. However, if the TPM implements any part of this, it MUST be implemented as specified.
2. If the TPM does not implement this section, it MUST NOT allow TPM_NV_INDEX_GPIO_00 to be defined. That is, calls to TPM_NV_DefineSpace with this index anywhere in the requested range MUST fail with the return code TPM_AREA_LOCKED.
3. Access to TPM_NV_INDEX_GPIO_00 MUST be treated as undefined (using the term "defined" in the TCG's technical context of NV Storage) until defined using TPM_NV_DefineSpace. While undefined, or if not implemented, the GPIO-Express-00 pin MUST be *high*.

The following text applies if and only if the TPM implements this GPIO feature.

4. If the TPM uses the recommended packaging in Section 14, it MUST assign the GPIO-Express-00 pin to the pin stated in that section. If the TPM does not use the recommended packaging, it must provide documentation to the platform manufacturer indicating which pin is assigned to GPIO-Express-00.
5. Between TPM_Init and TPM_Startup, the GPIO-Express-00 pin MAY be set to any value (i.e., it may be set to either *low*, *high*, or may float.) Upon completion of any TPM_Startup command, the GPIO-Express-00 pin MUST be set *high* and MUST not change until receipt of a TPM_NV_WriteValue* command.
6. When there is no TPM Owner, the TPM_NV_INDEX_GPIO_00 area MUST be marked as undefined.
7. The GPIO-Express-00 bit SHALL be bit 0 (i.e., the least significant bit) of TPM_NV_INDEX_GPIO_00. Bits 1 through 7 within TPM_NV_INDEX_GPIO_00 are not defined. Writes to bits 1-7 MUST be ignored. On read, bits 1-7 MUST be 0.
8. Upon a write to the GPIO-Express-00 bit, the TPM SHALL set the state of the GPIO-Express-00 pin to the value written to theGPIO-Express-00 bit. That is, writing a "1" to GPIO-Express-00 bit SHALL set the GPIO-Express-00 pin to *high*; writing a "0" to GPIO-Express-00 bit SHALL set the GPIO-Express-00 pin to *low*.
    a. The TPM MUST NOT change the state of the GPIO-Express-00 pin until a subsequent write to the GPIO-Express-00 bit, change of power state, or an operation that causes the TPM_NV_INDEX_GPIO_00 area to be undefined.
9. Upon a read from TPM_NV_INDEX_GPIO_00, the TPM MUST set the GPIO-Express-00 bit to the value of the GPIO-Express-00 pin at the time the TPM_NV_ReadValue* command is executed.

10. Electrical Requirements:

   a. The *low* and *high* values are as defined in the *PCI Electromechanical Card Specification* and referenced in the *LPC Interface Specification*.

   b. If the recommended packaging in Section 14 is used:

      i. The TPM MUST provide an internal weak pull-up resistor on the GPIO-Express-00 pin.

      ii. The GPIO-Express-00 MUST be an Open Collector.

   c. If the recommended packaging in Section 14 is not used, the TPM manufacturer MUST provide documentation to the platform manufacturer indicating the GPIO-Express-00 pin's electrical characteristics.

7. PCR Requirements

Start of informative comment

This section specifies the number and attributes for the set of PCRs. The purpose for specifying this is to establish a common and expected behavior for both hardware and software.

There needs to be an indicator that a T/OS is currently invoked regardless of whether the T/OS is currently controlling the platform. This indication is done using the TPM_STANY_FLAGS.T/OSPresent flag. The value of this flag upon TPM_Init is FALSE. Since the first D-CRTM, which begins the chain of trust for the T/OS, is signaled using the TPM_HASH_START command, this command is used to signal the presence of the T/OS by setting the TPM_STANY_FLAGS.TOSPresent flag to TRUE. When the T/OS exits (i.e., tears itself down, not just a change in control of the untrusted operating system), it must set this flag back to FALSE.

The TPM_STANY_FLAGS.TOSPresent flag is used to alter the behavior of the resetable PCRs to allow sealing and attestation to distinguish. This is done to distinguish between values extended into the resetable PCRs while the T/OS is launched and present and values extended to the resetable PCRs while there is not T/OS present. In this way, if an entity were able to extend a known good set of values that would indicate the presence of a T/OS, the values would still not be correct because the starting values are different between T/OS present and not present. Thus, if this flag is FALSE (i.e., no T/OS), the default value of TPM_PCRVALUE is -1.

End of informative comment

7.1 Number of PCRs

A conformant TPM MUST provide a minimum of 24 PCRs.

7.2 PCR Attributes

The attributes of the PCRs are to be as listed in Table 4.

280 **Table 4: PCR Attributes**

PCR Index	Alias	pcrReset	pcrResetLocal for Locality 4, 3, 2, 1, 0	pcrExtendLocal for Locality 4, 3 ,2, 1, 0
0 – 15	Static RTM	0	0,0,0,0,0	1,1,1,1,1
16	Debug	1	1,1,1,1,1	1,1,1,1,1
17	Locality 4	1	1,0,0,0,0	1,1,1,0,0[2]
18	Locality 3	1	1,0,0,0,0	1,1,1,0,0
19	Locality 2	1	1,0,0,0,0	0,1,1,0,0
20	Locality 1	1	1,0,1,0,0	0,1,1,1,0
21	T/OS Controlled	1	0,0,1,0,0	0,0,1,0,0
22	T/OS Controlled	1	0,0,1,0,0	0,0,1,0,0
23	Application Specific	1	1,1,1,1,1	1,1,1,1,1

1. The TPM MUST enforce the access to pcrResetLocal and pcrExtendLocal to those granted to each locality per Table 4.

For the remainder of this specification, the aliases for the PCRs will be use for consistency.

7.3 Reset Values

285 The contents of the cells in Table 5 is the value of the PCR at the conclusion of the state named in the header of the column.

Table 5: PCR Reset Values

PCR Index	TPM_Init	PCRReset T/OSPresent=FALSE	TPM_HASH_START	PCRReset T/OSPresent=TRUE
0-15	0	NC*	NC*	NC*
16	0	0	NC	0
17-22	-1*	-1*	0	0
23	0	0	NC	0

Table Notes:

NC =	No Change caused by the event.
NC* =	Though indicated as "No Change", it is the PCR attributes that prevent action.
-1 =	-1 is 20 bytes with all bits set to "1".

290

[2] See Section 7.4 for exception and restrictions to this behavior.

7.4 Restriction of Extend Behavior

Start of informative comment

The Locality 4 PCR contains the first measurement of the Dynamic RTM for the T/OS. The initial measurement performed by the RTM must be done using the TPM_HASH_* sequence. However, some environments may allow an extend operation to Locality 4 before executing the TPM_HASH_* sequence either during the launch sequence or from within a T/OS. This exposes a vulnerability in these environments where malicious code could launch in Locality 2 prior to the Dynamic RTM's execution of the TPM_HASH_* sequence, thus allowing the malicious code to perform a TPM_Extend operation with a value of its selection.

End of informative comment

1. When the Locality 4 PCR is at its default value, any operation that extends the Locality 4 PCR MUST originate from Locality 4.
2. The entry for the Locality 4 PCR in Section 7.2 SHALL be interpreted as if the column labeled "pcrExtendLocal for Locality 4, 3, 2, 1, 0" contains the bits: 1, 0 ,0 ,0, 0.
3. Once the Locality 4 PCR is no longer at its default state, Table 4 in Section 7.2 applies as written.

7.5 TPM Behavior for PCR Structure Values

Start of informative comment

The PCR information structures have several formats. This is mostly caused by the need to expand the information contained within the structure while maintaining backward compatibility with previous versions of TPMs. To provide consistent behavior, the following rules are specified.

End of informative comment

The following rules apply to the field TPM_PCR_SELECTION->sizeOfSelect in the structures TPM_PCR_INFO, TPM_PCR_INFO_SHORT, or TPM_PCR_INFO_LONG. The rules are ordered such that processing of the rules continues down, in the listed sequence, until a violation occurs. If no violation occurs, the TPM_PCR_SELECTION->sizeOfSelect value is legal.

If the TPM receives an illegal value for TPM_PCR_SELECTION->sizeOfSelect, it MUST error the command with the return code TPM_INVALID_PCR_INFO. The TPM MUST adhere to these rules when returning any of these structures to the software.

Size restrictions:

1. sizeOfSelect MUST NOT be 0 or 1.
2. When used in TPM_PCR_INFO_SHORT or TPM_PCR_INFO_LONG, sizeOfSelect MUST NOT be 2.

Optimization:

1. sizeOfSelect (and the corresponding PCRSelect field) MUST be the smallest possible value necessary to represent the selected PCRs; i.e., a structure with the leading byte(s) of the PCRSelect field that contain all zeros is illegal.

Exception to above sets of rules:

1. Within TPM_Quote, sizeOfSelect MUST allow the value 2.
   a. Within TPM_PCR_Reset, sizeOfSelect MUST allow the value 3.

8. Locality-Controlled Functions

8.1 Execution Sequence

Start of informative comment

Each RTM is provided a method to send component measurements. The Static RTM sends its measurements to the TPM using the TPM_SHA1* commands sent under Locality 0. Because the Dynamic RTM is started while the platform may be in an untrusted state, special and trusted mechanisms must be established to communicate the source of the corresponding commands to the TPM. These commands are indicated and controlled by the appropriate locality.

Locality 4 has the unique ability to send data to the TPM to be hashed and extended to the Locality 4 PCR. This is simply a stream of data sent within the TPM_HASH_DATA LPC command. There is no header or other information that accompanies the data. Upon receipt of the TPM_HASH_END LPC command, the TPM must complete the Hash and move the resultant value into the Locality 4 PCR.

The TPM_HASH_START / TPM_HASH_END sequence is also intended to act as a signal to the TPM to reset the resetable PCRs. The goal should be to insert as few LWAIT cycles as possible. If the TPM can reset the resetable PCR on TPM_HASH_START and do it quickly, then it can use TPM_HASH_START. If the TPM cannot, it should reset the resetable PCRs after the TPM_HASH_END LPC command is sent while it is calculating the hash of the received data. Either method is acceptable.

The data sent on the TPM_HASH_START and TPM_HASH_END has no meaning and may be any value.

End of informative comment

1. The TPM MUST perform the hash functions using the SHA-1 algorithm as defined by US FIPS 180-1.
2. Upon receipt of the TPM_HASH_START LPC command, the TPM MUST perform operations to affect the resetable PCRs per the following pseudo-code:

(Note: While the resulting functionality presented by the steps below are normative, the actual operations and their sequence as presented here are informative. There is no requirement to perform the following operations exactly as shown. However implemented though, the results MUST be the same as if the TPM were implemented as described below.)

3. Upon receipt of the TPM_HASH_START LPC command:
    a. If the TPM_ACCESS_x.activeLocality is not set, the TPM MUST set the TPM_ACCESS_x.activeLocality to indicate Locality 4. Any currently executing command MUST be aborted per and subject to Section 11.3.3.
    b. If TPM_ACCESS_x.activeLocality is set, and if the TPM_ACCESS_x.activeLocality is not 4, the TPM MUST ignore this command.
    c. The TPM MUST clear the write FIFO.
    d. If there is an exclusive transport session, it MUST be invalidated.
    e. Clear the TPM_ACCESS_x.tpmEstablishment bit to FALSE (0).

f. Set the TPM_STANY_FLAGS.TOSPresent flag to TRUE (1).

g. Set PCRs per column labeled TPM_HASH_START in Section 7.3.

h. Ignore any data component of the TPM_HASH_START LPC command.

i. Allocate tempLoc of a size required to perform the SHA-1 operation.

j. Initialize tempLoc per SHA-1.

4. Upon receipt of TPM_HASH_DATA LPC commands, transform tempLoc per SHA-1 with data received from this command:

   Repeat for each TPM_HASH_DATA LPC command received.

5. Upon receipt of a TPM_HASH_END LPC command:

   a. Ignore any data sent with the command.

   b. Perform finalize operation on tempLoc per SHA-1.

   c. Perform an "extend" operation of the value within tempLoc into the Locality 4 PCR.

      (1) That is, Locality 4 PCR = SHA-1([Locality 4 PCR] || tempLoc)

      (2) Note the following:

         (a) In the final step above, [Locality 4 PCR] within and before the SHA-1 function is TPM_PCRVALUE = 0 (i.e., 20 bytes of all zeros).

         (b) If no TPM_HASH_DATA LPC command is sent between the TPM_HASH_START and TPM_HASH_END LPC commands, tempLoc will contain the initialization value as defined by SHA-1 with no transformations into it. Thus, in this case, the contents of the Locality 4 PCR SHALL be:

            (i) Locality 4 PCR = SHA-1 (0 || (SHA-1 defined initialization value))

   d. Clear TPM_ACCESS_x.activeLocality for Locality 4.

6. If there was not a TPM_HASH_START or TPM_HASH_DATA LPC command prior to receipt of the TPM_HASH_END, the TPM MUST NOT change the state of the TPM but MUST clear TPM_ACCESS_x.activeLocality for Locality 4.

7. After receipt of a TPM_HASH_START LPC command:

   a. All data sent to the TPM_DATA_FIFO_4 MUST be treated as data for the hash function.

   b. All cycles and commands other than a write to the TPM_HASH_DATA and TPM_HASH_END LPC commands MUST be ignored until a TPM_HASH_END command is received.

8. If a TPM_PCR_Reset command is received with the correct Locality even after the TPM_HASH_END LPC command, the TPM MUST reset the indicated PCRs as indicated in Section 7.3 Reset Values.

9. There is no response packet returned as a result of any of the TPM_HASH_* LPC commands.

8.2 Timing and Protocol

Start of informative comment

The D-CRTM executes within a resource-restricted environment which is among the reasons the TPM_HASH_* protocol is used rather than the more obvious TPM command ordinals (e.g., TPM_Extend). It is also difficult and unnecessary for this environment to use the register-based TPM_STS_x protocols. Therefore, during the Locality 4 TPM_HASH_* commands, the only method used to throttle commands to the TPM is using the LPC bus. (There is no data returning from TPM within this environment) Specifically, the TPM uses the LPC bus “long wait” cycle to indicate to the “host” (using LPC terms) that it is not able to accept more data.

This environment also may not be conducive to “timeouts” and may be very susceptible to delays or hangs. It is important that the TPM be designed to not create excessive delays or cause the LPC bus to hang during this time.

End of informative comment

1. During the TPM_HASH_* commands, the TPM MUST use the LPC bus “long wait cycle” to indicate its inability to accept more commands or data. While the TPM MAY set the TPM_STS_x.* status bits they are “undefined” during these commands and will likely not be read and will not be honored.
2. The TPM MUST respond to the TPM_HASH_START command within TIMEOUT_B.
3. The TPM SHOULD respond to each TPM_HASH_DATA and TPM_HASH_END command within 250 microseconds and MUST respond within TIMEOUT_B.

9. Locality

9.1 TPM Locality Levels

Start of informative comment

There are six levels of locality in the TPM: Locality Legacy, Locality 0 (No locality), Locality 1-4. The usages of these are defined in the *PC Client Implementation Specification* (q.v.).

Note that the TPM must support the last five levels of Locality, 0 - 4 above, to be compliant with TCG 1.2. To be backwards compatible with TCG 1.1 software and platforms, the TPM must support the Legacy Locality also.

Each PCR, during manufacturing the TPM, has the locality level set for four types of operations: reset, read, use (e.g., TPM_Unseal, etc.) or modification (e.g., EXTEND). The generic TPM definition allows for the specification of four locality modifiers. The present definition defines two of these modifiers and leaves the other two undefined. Each PCR has a 4 bit bit-mask. These bits define the modifier which must be present to allow the operation. The bit mask is selective. That is, the setting of bit 2 does not imply bit 1. If the PCR wants to allow for both Locality 1 and Locality 2, both bits must be set in the mask. If a command attempts an operation using the PCR, it must be received at the TPM with the appropriate modifier. Setting bit 1 does not imply bit 2 is set, even though Locality 2 is a "higher" locality. An example of the use of locality is that Locality PCRs 1 through 4 must be programmed so that they will only be reset from Locality 4, and only extended if TPM_Extend is done with Locality 2.

TPM commands may also require the locality modifier. The TPM, upon receipt of each command, sets the locality modifier for the command. The modifier, as presently defined, may be 0 through 4.

The platform-specific specification indicates what the locality settings are for each PCR register. PCR[0-15] do not allow the reset and have no locality requirements (this matches the usage model of these PCR registers for TCG 1.1. PCR[17-20] can be reset using Locality 4 and can be read, used or modified using Locality 1-3. PCR[21-22] are reserved for and controlled by the T/OS. PCR[23]'s usage is reserved and must not be used.

For the platform, the locality level is indicated by the address used along with the new LPC Start cycle. The new LPC transaction has a 16-bit address. Table 6 shows the locality based on the 16-bit address. Note that we have also defined a new LPC bus cycle to communicate with the TPM. This was done to prevent simple hardware attacks using a device on the LPC bus that decoded I/O or memory cycles using the previously defined START field. Cycles using the normal memory read/write or I/O read/write START field to the following ranges are not decoded by the TPM.

End of informative comment

Table 6: Locality Address Definitions

System/Software Address	LPC Address (Using a New LPC START Cycle)	Locality
FED4_0xxxh	0xxxh	0
FED4_1xxxh	1xxxh	1
FED4_2xxxh	2xxxh	2
FED4_3xxxh	3xxxh	3
FED4_4xxxh	4xxxh	4

Start of informative comment

There are also legacy I/O cycles that TPM 1.1 used. The TPM 1.2 may continue to use legacy I/O cycles. The TPM must not respond to I/O cycles when any of the TPM_ACCESS_x.activeLocality bits is set. It is possible that code could be written to provide some software handshake to allow legacy locality and other localities to coexist, but that is not the intended usage. It is expected that a platform may run a particular piece of code that uses legacy locality when there is no other code in the platform that accesses the TPM. Any platform that has multiple pieces of code accessing the TPM should use Locality 0-3 and not use legacy locality.

If the TPM has been used by Locality 0-4 transactions, and then placed in the "free" state whereby no TPM_ACCESS_x.activeLocality bit is set, the TPM can accept legacy I/O cycles as long as no other software sets any If TPM_ACCESS_x.activeLocality bit. The TPM hardware does not need to "remember" that it was used by locality-based software. It simply accepts legacy I/O cycles whenever no TPM_ACCESS_x.activeLocality bit is set.

End of informative comment

1. The TPM MUST support five levels of locality (Locality 0, Locality 1, Locality 2, Locality 3, and Locality 4) to be conformant with the TCG 1.2 specification. To be backwards compatible with TCG 1.1 software and platforms, the TPM MAY also support the legacy addresses.
2. Commands sent to the TPM using Locality 0-4 MUST use the LPC Locality Cycles as specified in Section 13.2. Commands sent using legacy addressing MUST use the standard LPC bus cycles and MUST NOT use the LPC Locality Cycles.
3. Each PCR has a bit mask of 4 bits that allows specification of which modifier MUST be present to allow the operation.
4. If a command attempts an operation using the PCR, it MUST be received at the TPM with the appropriate modifier.
5. TPM commands MAY also require the locality modifier.
6. The TPM 1.2 MAY continue to use legacy I/O cycles. The TPM MUST NOT respond to I/O cycles when any of the TPM_ACCESS_x.activeLocality bits are set.
7. Any platform that has multiple pieces of code that access the TPM SHOULD use Locality 0-3 for that and SHOULD NOT use legacy locality.
8. PCR register modifications, uses, or commands that require locality (i.e., Locality 1-4) MUST NOT execute when presented using Locality 0.

9. The Locality 4 modifier CANNOT be generated by any software including the T/OS. It must be generated by the Dynamic RTM.
10. For the resetable PCRs, the TPM_PCR_RESET command MUST require the Locality 4 modifier in order to be executed. The HASH functionality is done with Locality 4, since it is generated by the Dynamic RTM.

9.2 Locality Uses

Start of informative comment

The idea behind locality is that certain combinations of software and hardware are allowed more privileges than other combinations. For instance, the highest level of locality might be cycles that only hardware could create. Since the hardware is virus-proof (at least in theory), it has a higher level of trust than cycles code generates.

Locality 4 is the highest level defined. These cycles are generated by hardware. This would include things such as the TPM_PCR_Reset and TPM_HASH_* LPC commands. The hardware guarantees that the TPM_PCR_Reset or HASH operations occurred, and that the correct data was sent.

Next, assume a platform that has software based on either using no PCRs or the set of resetable PCRs (the operating system based on the Static RTM or Static Operating System) and trusted software: a T/OS. In this case, there is a need to differentiate cycles from the operating systems. Localities 1-3 are used by the T/OS for its transactions. The trusted system has created certain values in the new resetable PCRs. Only trusted software should be able to issue commands based on those PCRs. This software is Locality 1-3 software.

The non-resetable PCRs (i.e., PCR[0-15]) are not part of the T/OS domain, so Locality 0 transactions can freely use those PCRs, but must be prevented from using the resetable PCRs.

The platform needs some way to differentiate the code executing in the Static Operating System from the code to establish and run the T/OS. The method defined for the TPM is address based. The system can prevent certain agents from accessing certain ranges, thus preventing Static Operating System code from issuing Locality 1-4 commands.

One concern regarding locality is that once the resetable PCRs have been loaded, the platform should prevent Locality 0 accesses from using these PCRs. It is acceptable for a system to define all accesses to be Locality 1-4 and not use Locality 0, if that system is not concerned with legacy or software based on the Static RTM. If other transport mechanisms are sufficient to provide the correct protections, Locality 0 need not be used.

End of informative comment

Usage of the Locality 0 PCRs is determined by the *TCG PC Client Specific Implementation Specification.*

9.3 Locality Usage per Register

Start of informative comment

Table 7 shows which cycles must be accepted by TPM. The following rules apply:

An Abort requires that the cycle have no affect on the TPM. There are two possible implementations, either of which is acceptable:

1. The TPM may simply not drive a valid SYNC. When it sees a cycle it should abort the command. This will return FFh to the CPU for reads; writes are dropped.
2. For writes, the TPM may accept them and do nothing with the writes. If the TPM claims the write on the LPC bus, it must guarantee there are no affects on the TPM. The TPM will not provide any TPM-Response to these writes, nor does it provide any indication that it has seen a write but dropped it. For reads, the TPM, if it responds with a SYNC, must return FFh as the data. This mimics a true LPC or PCI Master Abort from the CPU's perspective.

End of informative comment

1. If TPM_ACCESS_x.activeLocality setting changes when a command is executing (TPM_STS_x.commandReady is cleared), the TPM SHOULD abort the currently executing command.
2. An Abort causes the TPM to ignore the current LPC cycle.

Table 7: Register Usage Based on Locality Setting

Register	TPM_ACCESS_x.activeLocality Set for ThisLocality		TPM_ACCESS_x.activeLocality Set for Some Other Locality		No TPM_ACCESS_x.activeLocality Set	
	READ	WRITE	READ	WRITE	READ	WRITE
TPM_STS_x	TPM returns correct status	Status bits updated	Abort	Abort	Abort	Abort
TPM_INT_ENABLE_x	TPM returns correct status	Bits updated	TPM returns correct value	Abort	TPM returns correct value	Abort
TPM_INT_VECTOR_x	TPM returns correct status	Bits updated	TPM returns correct value	Abort	TPM returns correct value	Abort
TPM_INT_STATUS_x	TPM returns correct status	Interrupt cleared	TPM returns correct value	Abort	TPM returns correct value	Abort
TPM_INTF_CAPABILITY_x	TPM returns correct information	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register
TPM_ACCESS_x	TPM returns correct value	Writes accepted	TPM returns correct value	Writes are accepted	TPM returns correct value	Writes are accepted
Read TPM_DATA_FIFO_x	TPM returns correct data	Abort	Abort	Abort	Abort	Abort
Write TPM_DATA_FIFO_x	Abort	TPM accepts data and command	Abort	Abort	Abort	Abort
Configuration registers– 0F00h to 0FFFh	TPM returns correct information	Status bits updated	TPM returns correct information	Abort	TPM returns correct information	Abort

Register	TPM_ACCESS_x.activeLocality Set for ThisLocality		TPM_ACCESS_x.activeLocality Set for Some Other Locality		No TPM_ACCESS_x.activeLocality Set	
	READ	WRITE	READ	WRITE	READ	WRITE
TPM_HASH_START	Abort (HASH is write)	TPM accepts	Abort	Abort	Abort	TPM accepts (and sets TPM_ACCESS_x .activeLocality for Locality 4)
TPM_HASH_DATA	Abort	TPM accepts	Abort	Abort	Abort	Abort
TPM_HASH_END	Abort	TPM accepts and clears TPM_ACCESS_x .activeLocality for Locality 4	Abort	Abort	Abort	Abort

9.4 TPM Legacy I/O Space and TPM 1.2 Memory Mapped Space

Start of informative comment

Typically, TPMs designed to work with *TCG Main Specification, Version 1.1b* used addressing that is described in this section as legacy addressing. In the *TCG Main Specification, Version 1.1b* there was no mandate or recommendation regarding addressing. This version of the specification does mandate a specific addressing scheme, however, it is recognized that some environments may be required to provide legacy addressing to allow backward compatibility. The new addressing scheme takes into account the set of known addresses used. This address mapping scheme allows a TPM 1.2 to be used in an environment where compatibility with a TPM 1.1b is needed.

The 1.1 TPM has traditionally been located in I/O space. It has used addresses 2Eh/2Fh, 4Eh/4Fh, and others. Devices that have fixed addresses in the legacy I/O space many times conflict with other devices that have fixed addresses. Therefore, the following is proposed as a mapping of legacy registers into memory space.

The TPM would accept either an I/O access to 2Eh, 4Eh, or 7Eh (depending on the TPM and its strapping) or a TPM-Read or TPM-Write to FED4_0F80h as an access to the Legacy 0 register. Note that the registers have been extended to 16 bits with the addition of the Legacy1b and Legacy2b locations. Note also that this is not a proposal to have separate registers, at I/O 2Eh and 0F80h, but one physical register that may be addressed in two different ways.

The TPM 1.2 may be used in chipsets that do not implement the memory-mapped to TPM-Read or TPM-Write cycles. For these systems, the legacy I/O space or some other mechanism must be used to access the TPM.

End of informative comment

1. The TPM 1.2 MAY be used in chipsets that do not implement the memory-mapped to TPM-Read or TPM-Write cycles. For these systems, the legacy I/O space or some other mechanism MUST be used to access the TPM.
2. If TPM legacy I/O space is used, it MUST only be used and allowed while the TPM's locality is not set.

Table 8 shows the mapping that MAY be implemented to support legacy addressing.

Table 8: Legacy Port Usage

Name	Use	Access	I/O Address	System Memory Address	TPM 16-bit Address Using the New LPC TPM Cycles	Usage
Legacy1	First legacy address	Read/Write	2Eh, 4Eh, 7Eh, EEh, or other	FED4_0F80h	0F80h	Vendor defined
Legacy1b	Not used today	Read/Write	Additional 8 bits for Legacy1 register	FED4_0F84h	0F84h	Vendor defined
Legacy2	Second legacy address	Read/Write	2Fh,r 4Fh, 7Fh, EFh, or other	FED4_0F88h	0F88h	Vendor defined
Legacy2b	Not used today	Read/Write	Additional 8 bits for Legacy2 register	FED4_0F8Ch	0F8Ch	Vendor defined

Addresses not listed above but that are in the 0F80h to 0F8Fh range are reserved and MAY return all 0's or may ignore the least significant 2 bits of the address and alias to a legitimate address. For instance, 0F86h can return all 0's or alias to 0F84h and return the data at that location.

9.5 TPM Register Space Decode

Start of informative comment

Many of the registers in the TPM Register Space contain an address range. Most of these registers are accessed with the TPM decoding all addresses within the specified address ranges.

However, one of these registers is the TPM data register (TPM_DATA_FIFO_x) which is defined as having four addresses. For this register, the addresses within this range are aliased to one internal register.

The LPC bus transfers a single byte per transaction to any of the registers within this range. The chipset may, for performance reasons, send a DW (4 bytes) for each transaction. This will appear as four distinct LPC bus transactions, with each incrementing the address by 1 with each set of transactions starting with the least significant byte – thus being little-endian. Because the TPM can accept only 1 byte per transaction, the TPM must ignore the 2 least significant bits of the address for the data registers thus receiving the data serially. However, it should not require or expect that each address is incremented modulo-4.

End of informative comment

1. TPM_DATA_FIFO_x
   The TPM MUST ignore the 2 least significant bits of the address for these registers and accept each byte received to any of the addresses within this register as a single transfer to be the base address.
2. All other registers:
   a. MUST do a full decode down to the byte level (unless otherwise specified).
   b. That have multiple addresses, the lowest address within the set of addresses MUST contain the least significant bits with the bits incrementing to each successive

address up to the highest address that MUST contain the most significant bits. For example, the TPM_INT_ENABLE_0 register for Locality 0 MUST be implemented as shown in Table 9.

Table 9: Example Bit-to-Address Mapping

Address	Data Bit Position
0008	7:0
0009	15:8
000A	23:16
000B	31:24

10. TPM Register Space

Table 10 lists the addresses decoded by the TPM. Note that only the TPM_ACCESS_x register has multiple copies, one per locality. The other addresses alias to a single register with the locality used to determine if accesses are permitted or Aborted.

625 **Note:** that TPM_DATA_FIFO_x is used for both reading and writing. When referring to reading the status, it may be called the ReadFIFO; when writing the command it may be called the WriteFIFO. For Locality 4, the TPM_HASH_DATA port is the same as the TPM_DATA_FIFO_x.

Table 10: Allocation of Register Space for TPM Access

Offset	Register Name	Description
Locality 0		
0000h	TPM_ACCESS_0	Used to gain ownership for this particular port.
000Bh-0008h	TPM_INT_ENABLE_0	Controls interrupts
000Ch	TPM_INT_VECTOR_0	SIRQ vector to be used by the TPM
0013h-0010h	TPM_INT_STATUS_0	What caused interrupt
0017h-0014h	TPM_INTF_CAPABILITY_0	Shows which interrupts are supported by that particular TPM
001Ah-0018h	TPM_STS_0	Status Register. Provides status of the TPM
00027h-0024h	TPM_DATA_FIFO_0	Read or write FIFO, depending on transaction. Between TPM_HASH_START and TPM_HASH_END, writes to this range are aborted and have no affect on the part. These four addresses are aliased to one inside the TPM. The TPM is not required to check that the addresses on the LPC bus are incrementing modulo-4, even though platform hardware would most likely send it that way. The read or write data could be performed by accessing 0024h repeatedly without using the other addresses.
0F03h-0F00h	TPM_DID_VID_0	Vendor and device ID
0F04h	TPM_RID_0	Revision ID
0F7Fh-0F05h		TCG defined configuration registers
0F80h	FIRST_LEGACY_ADDRESS_0	Alias to I/O legacy space
0F84h	FIRST_LEGACY_ADDRESS_EXTENSION_0	Additional 8 bits for I/O legacy space extension
0F88h	SECOND_LEGACY_ADDRESS_0	Alias to second I/O legacy space
0F8Ch	SECOND_LEGACY_ADDRESS_EXTENSION_0	Additional 8 bits for second I/O legacy space extension
0FFFh-0F90h		Vendor-defined configuration registers
Locality 1		
1000h	TPM_ACCESS_1	Same as TPM_ACCESS_0
100Bh-1008h	TPM_INT_ENABLE_1	Same as TPM_INT_ENABLE_0
100Ch	TPM_INT_VECTOR_1	Same as TPM_INT_VECTOR_0
1013h-1010h	TPM_INT_STATUS_1	Same as TPM_INT_STATUS_0
1017h-1014h	TPM_INTF_CAPABILITY_1	Same as TPM_INTF_CAPABILITY_0
101Ah-1018h	TPM_STS_1	Same as TPM_STS_0
1027h-1024h	TPM_DATA_FIFO_1	Same as TPM_DATA_FIFO_0
1F03h-1F00h	TPM_DID_VID_1	Same as TPM_DID_VID_0
1F04h	TPM_RID_1	Same as TPM_RID_0

Offset	Register Name	Description
1F7Fh-1F05h		TCG defined configuration registers
1F80h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_0 but MUST NOT be used
1F84h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
1F88h	Reserved	Parallel to SECOND_LEGACY_ADDRESS_0 but MUST NOT be used
1F8Ch	Reserved	Parallel to SECOND_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
1FFFh-1F90h		Vendor-defined configuration registers
Locality 2		
2000h	TPM_ACCESS_2	Same as TPM_ACCESS_0
200Bh-2008h	TPM_INT_ENABLE_2	Same as TPM_INT_ENABLE_0
200Ch	TPM_INT_VECTOR_2	Same as TPM_INT_VECTOR_0
2013h-2010h	TPM_INT_STATUS_2	Same as TPM_INT_STATUS_0
2017h-2014h	TPM_INTF_CAPABILITY_2	Same as TPM_INTF_CAPABILITY_0
201Ah-2018h	TPM_STS_2	Same as TPM_STS_0
2027h-2024h	TPM_DATA_FIFO_2	Same as TPM_DATA_FIFO_0
2F03h-2F00h	TPM_DID_VID_2	Same as TPM_DID_VID_0
2F04h	TPM_RID_2	Same as TPM_RID_0
2F7Fh-2F05h		TCG defined configuration registers
2F80h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_0 but MUST NOT be used
2F84h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
2F88h	Reserved	Parallel to SECOND_LEGACY_ADDRESS_0 but MUST NOT be used
2F8Ch	Reserved	Parallel to SECOND_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
2FFFh-2F90h		Vendor-defined configuration registers
Locality 3		
3000h	TPM_ACCESS_3	Same as TPM_ACCESS_0
300Bh-30008h	TPM_INT_ENABLE_3	Same as TPM_INT_ENABLE_0
300Ch	TPM_INT_VECTOR_3	Same as TPM_INT_VECTOR_0
3013h-3010h	TPM_INT_STATUS_3	Same as TPM_INT_STATUS_0
3017h-3014h	TPM_INTF_CAPABILITY_3	Same as TPM_INTF_CAPABILITY_0
301Ah-3018h	TPM_STS_3	Same as TPM_STS_0
3027h-3024h	TPM_DATA_FIFO_3	Same as TPM_DATA_FIFO_0
3F03h-3F00h	TPM_DID_VID_3	Same as TPM_DID_VID_0
3F04h	TPM_RID_3	Same as TPM_RID_0
3F7Fh-3F05h		TCG defined configuration registers
3F80h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_0 but MUST NOT be used
3F84h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used

Offset	Register Name	Description
3F88h	Reserved	Parallel to SECOND_LEGACY_ADDRESS_0 but MUST NOT be used
3F8Ch	Reserved	Parallel to SECOND_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
3FFFh-3F90h		Vendor-defined configuration registers
Locality 4		
4000h	TPM_ACCESS_4	Same as TPM_ACCESS_0
400Bh-4008h	TPM_INT_ENABLE_4	Same as TPM_INT_ENABLE_0
400Ch	TPM_INT_VECTOR_4	Same as TPM_INT_VECTOR_0
4013h-4010h	TPM_INT_STATUS_4	Same as TPM_INT_STATUS_0
4017h-4014h	TPM_INTF_CAPABILITY_4	Same as TPM_INTF_CAPABILITY_0
401Ah-4018h	TPM_STS_4	Same as TPM_STS_0
4020h	TPM_HASH_END	This signals the end of the hash operation. When the TPM receives this command and has finished hashing all data received on the TPM_HASH_DATA port, it MUST write the final hash into the Locality 4 PCR. This command MUST be done on the LPC bus as a single write to 4020h. Writes to 4021h to 4023h are not decoded by the TPM. TPM_HASH_END is an implicit release of the TPM. This is equivalent to writing TPM_ACCESS_4.activeLocality.
4027h-4024h	TPM_HASH_DATA/ TPM_DATA_FIFO_4	This port is used to send data that the TPM is to hash. There is no TCG style command header or format associated with this port. Once the TPM_HASH_START command is received, the TPM MUST take all data received on the LPC bus that falls in the range of these four addresses and do a hash on it. Between TPM_HASH_START and TPM_HASH_END, writes to this register are treated as data and part of the Hashed value. Outside that window, writes to this register are treated as part of a TPM command. These four addresses are aliased to one inside the TPM. The TPM is not required to check that the addresses on the LPC bus are incrementing modulo-4, even though platform hardware would most likely send it that way. The HASH could be done by writing to 4024h repeatedly without using the other addresses.
4028h	TPM_HASH_START	This signals the start of the hash operation. This command MUST be done on the LPC bus as a single write to 4028h. Writes to 4029h to 402Bh are not decoded by TPM.
4F03h-4F00h	TPM_DID_VID_4	Same as TPM_DID_VID_0
4F04h	TPM_RID_4	Same as TPM_RID_0
4F7Fh-4F05h		TCG defined configuration registers
4F80h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_0 but MUST NOT be used
4F84h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
4F88h	Reserved	Parallel to SECOND_LEGACY_ADDRESS_0 but MUST NOT be used
4F8Ch	Reserved	Parallel to SECOND_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
4FFFh-4F90h		Vendor-defined configuration registers

Offset	Register Name	Description
All addresses not defined in the table above		Reserved, reads return FFh; writes are dropped.

The following sections provide implementation details on the defined registers. Note that each register has multiple versions; e.g., TPM_STS_x represents TPM_STS_0, TPM_STS_1, TPM_STS_2, TPM_STS_3, and TPM_STS_4.

Note that the DID/VID, RID, and all the TCG and vendor-specific registers have only one physical copy, but can be accessed from any locality. The configuration registers can be read by any locality at any time but can only be written by the active locality.

11. System Interaction and Flows

11.1 Startup Command

Start of informative comment

1. The TPM must have a startup command before the first command. The Extend command does not work until after Startup command. The optimal place to perform the TPM_Startup command is in the BIOS. After the BIOS hands over control to an operating system, it can determine if the TPM_Startup has been executed and, if necessary, perform the operation itself.
2. The TPM requires that a TPM_Startup command be done before TPM_HASH_START is accepted. Software must issue the STARTUP command. If TPM_HASH_START is done before TPM_Startup, it is an error.

End of informative comment

1. The TPM MUST have a startup command before the first extend or hash command.
2. The TPM requires that a TPM_Startup command be done before TPM_HASH_START is accepted. If TPM_HASH_START is received by the TPM prior to receiving the TPM_Startup command, the Locality 1-4 PCRs MUST remain in their default state. The TPM MAY remain in either the Locality 0 or Locality 4 state, but MUST NOT enter any other locality state. The TPM MUST enter the same operational state as when a normal TPM command was issued prior to the TPM_Startup command.

11.2 Configuration Registers

11.2.1 DID/VID Register

Table 11: DID/VID Register

Abbreviation:			TPM_DID_VID_x
General Description:			Vendor and Device ID for the TPM
Default			Vendor specific
Bit Descriptions:			
31:16	Read Only	did	Device ID – vendor specific
15:0	Read Only	vid	Vendor ID – Assigned by TCG Administrator. This is represented within the register in big-endian format. For example, a vendor ID of 0x1234 would be represented as: Bits 7:0 = 34 (0011 0100); Bits 15:8 = 12 (0001 0010).

11.2.2 RID Register

Table 12: RID Register

Abbreviation:	TPM_RID_x		
General Description:	Revision ID for the TPM		
Default	Specific to each revision		
Bit Descriptions:			
7:0	Read Only	rid	Revision ID – specifies the revision of the component

11.2.3 Legacy Configuration Register

Table 13: Legacy Configuration Register

Abbreviation:			
General Description:	Alias of the standard I/O configuration		
Default	Specific to each vendor		
Bit Descriptions:			
7:0	R/W		Configuration space defined by each vendor. The TPM MUST support one of the options presented here. It may not use the range of 0F80h to 0F8Fh for some other purpose. NOTE: The TPM MUST either alias the ranges of 0F80h, 0F84h, 0F88h, and 0F8Ch to the same registers defined in I/O space for that vendor's TPM or the TPM MUST abort cycles to these four addresses. NOTE: The TPM MUST also decode the range of XF80h to XF8Fh for X = 1 to 4 for the other localities exactly the way it does for Locality 0; however, these cycles MUST be aborted.

11.3 TPM's Software Interaction

Start of informative comment

Before the T/OS is started, legacy software can use the TPM. Even if the T/OS is never started, the TPM may be used by legacy applications. These legacy applications will use the TPM's legacy I/O-mapped port.

After the T/OS has been started, the TPM has two possible users:

1. Applications that do not use the T/OS to talk to the TPM.
2. Applications that use T/OS services to talk to the TPM.

Since there are two possible users of the TPM, some synchronization method is required.

Assume that the legacy code uses only the legacy I/O mapped port and that the T/OS will use the newly defined TPM-Read and TPM-Write cycles. Note that how the platform maps CPU accesses to these new cycles is chipset/platform dependent. This defines the TPM as having two logical ports, one I/O mapped and the other TPM mapped.

When there are multiple software users of the TPM, the following explains the handshake mechanism.

Each software agent, when it wishes to use the TPM, must write a "1" to TPM_ACCESS_x.requestUse. In this case, the TPM is idle so the first agent that writes its bit will become the user. The TPM must set TPM_ACCESS_x.activeLocality for the locality that gains access to TPM. All other localities that have written the TPM_ACCESS_x.requestUse bit must poll on the TPM_ACCESS_x register and read a “0” for TPM_ACCESS_x.activeLocality. The winning locality will read a “1” on this bit and start issuing commands to the TPM.

When the winning locality is finished with the TPM, it must write a “1” to its TPM_ACCESS_x.activeLocality. This indicates that it is finished with its series of commands. The TPM must look at all pending TPM_ACCESS_x.requestUse bits and grant the access to the highest locality with its bit set by setting TPM_ACCESS_x.activeLocality for that locality. The others continue waiting.

If software, for some reason, decides a lesser locality’s software is not playing fair or is hung (by exceeding the maximum timeout value as specified by the TSS), then it can seize the TPM from the present user, as long as that user is at a lower locality. To accomplish this, a “1” is written to the TPM_ACCESS_x.Seize register bit. This forces the TPM to stop honoring cycles from the other locality, and only honor the new locality’s requests.

End of informative comment

11.3.1 Handling Command FIFOs

Start of informative comment

Before issuing a command to the TPM, the software must check to see if it is in a state where it can accept commands. This is done by reading the TPM_STS_x register.

TPM_STS_x.burstCount field indicates if the command FIFO is ready to accept more data of a command. The TPM is not allowed to drop a cycle because of an internal stall. If the TPM cannot accept a write cycle, then it should stall the LPC bus using standard LPC wait syncs. TPM_STS_x.Expect indicates if the TPM is expecting more data for the command being sent.

Given that there may be a large amount of data to be written to the TPM, and that the LPC bus is slow relative to the CPU, there needs to be some software handshake that well-behaved software can use to know when it can send more data and when it should hold off. This allows the software to throttle itself so that other agents that lie along the same path as the CPU-to-TPM are not starved. Software can send some reasonable amount of data, say 64 bytes, and then poll until the TPM is empty before sending the next 64 bytes. However, under no circumstances may the TPM drop a cycle.

End of informative comment

1. Before issuing a command to the TPM, the software MUST check to see if it is in a state where it can accept commands; this is done by reading the TPM_STS_x register.
2. The TPM MUST NOT drop a cycle because of an internal stall. If the TPM cannot accept a write cycle, then it MUST stall the LPC bus using standard LPC wait syncs. TPM_STS_x.Expect indicates if the TPM is expecting more data for the command being sent.

11.3.2 Completion Command Details

11.3.2.1 Data Availability

Start of informative comment

When the command has been issued, the TPM must put the results in a ReadFIFO, which can be read via the TPM_DATA_FIFO_x register. The TPM_STS_x.dataAvail register indicates when the results of the command are available. After data is available, the TPM_STS_x.burstCount field is used to read the data.

End of informative comment

1. Before reading the results of a command from a ReadFIFO, the software SHOULD read the TPM_STS_x.dataAvail register to see if the data from the TPM is available. If the TPM_STS_x.dataAvail bit is "1", at least 1 byte of the completion return data is available. The TPM_STS_x.burstCount field is used to throttle bytes from the TPM.
    a. Any value previously read from the TPM_STS_x.burstCount field is invalidated following a write to TPM_STS_x.responseRetry or an Abort operation.
2. If the TPM_INTF_CAPABILITY_x.burstCountStatic indicates a *dynamic* TPM_STS_x.burstCount:
    a. The TPM_STS_x.burstCount indicates the number of bytes available in the ReadFIFO. This register does not provide a total count of bytes to be returned; rather, it only indicates how many bytes are currently available in the ReadFIFO.
        i. As data is read from the ReadFIFO, the value of this register is decremented.
        ii. As the TPM either completes the operation providing more data to the ReadFIFO or the TPM is replenishing data read from the ReadFIFO, this register is incremented.
3. If the TPM_INTF_CAPABILITY_x.burstCountStatic indicates a *static* TPM_STS_x.burstCount:
    a. The TPM_STS_x.burstCount indicates the number of bytes that could be read without incurring an LPC long wait cycle in the ReadFIFO. This register does not provide a total count of bytes to be returned, rather; it only indicates how many bytes are currently available in the ReadFIFO.

11.3.3 Aborts

Start of informative comment

There are several causes for a TPM to abort an executing command: TPM_ACCESS_x.Seize, TPM_STS_x.commandReady, and TPM_ACCESS_x.activeLocality. Because of implementation differences and the non-deterministic nature of some commands that may be executing, the TPM may not be able to respond to an abort either immediately nor with a predictable time. This non-deterministic behavior causes driver design difficulties because the driver will not be able to distinguish between a TPM waiting normally and a TPM that has errored and is not responsive. Therefore, a maximum amount of time is specified so the TPM manufacturers have a design parameter that the drivers can rely upon.

Note: Because there is no requirement for a TPM to handle more than one operation at a time, there can be no actual and standardized TPM command to cause an abort. The method for signaling an abort to the TPM is by writing to specific registers.

End of informative comment

1. Upon a successful abort, the TPM must stop the currently executing command, clear the FIFOs, and transition to either the ready or idle state.
2. The following operations will cause an abort:
   a. Writing a “1” to TPM_STS_x.commandReady during the execution of a command.
   b. Writing a “1” to TPM_STS_x.commandReady during the receipt of commands but before execution of a command.
   c. Writing a “1” to TPM_ACCESS_x.Seize bit but only when successful.
   d. Writing a “1” to TPM_ACCESS_x.activeLocality.
   e. Successful completion of the TPM_HASH_START per Section 8.

1. The TPM internal state MAY be either pre-aborted command or post-aborted command state and MUST not be any intermediate value.
2. For commands indicated as short or medium duration (i.e., those that do not cause a key generation), the TPM MUST respond to an abort in less than the medium duration as reported by TPM_GetCapability command. For command indicated as long duration or those that cause key generation the TPM MUST respond to a request to abort the command within 2 seconds.

11.3.4 Command Duration

Start of informative comment

It is important to distinguish between the two terms: duration and timeout. Duration is the amount of time for a TPM to execute a command once the TPM has received the command’s complete set of bytes and the software starts the operation by writing a “1” to TPM_STS_x.tpmGo. Duration has no relationship to the timings of the interface protocols. Those are timeouts (see Section 11.3.5).

During a platform’s bootstrapping phase, performance is critical and resources are scarce to compensate for differences in TPM implementations. For this reason, constraints are placed on commands that are typically required during this phase. Since the same platform requirements drive the reasons for making commands available before a selftest is complete, the commands listed in this section are similar to those in the Section 11.3.7. An addition is the command that retrieves the actual command duration: TPM GetCapability with the capability TPM_CAP_PROP_DURATION because until this command is called, the software has no way of knowing how long it will take.

Once the platform has completed its bootstrapping phase to the point where it has sufficient resources to deal with differing command durations (e.g., for proper user experience), it should call GetCapability: TPM_CAP_PROP_DURATION to get the actual values.

End of informative comment

1. Command Duration is defined as the time between the TPM's receipt of a "1" to TPM_STS_x.tpmGo and the TPM setting both TPM_STS_x.dataAvail and TPM_STS_x.stsValid bits to a "1".
2. The duration of the commands listed in Section 11.3.7 and the command TPM_GetCapability, with the capability TPM_CAP_PROP_DURATION, MUST be less than 750 ms.

11.3.5 Timeouts

Start of informative comment

The term timeout applies to timings between various states or transitions within the interface protocol. Timeout values are the purview of this specification and are not related to duration (see Section 11.3.4).

Because of the variations between implementations, it is not practical to specify timeout values that apply to all implementations. Efficient software must have the means to provide optimizations; therefore, software should be able to determine, with some level of granularity, when a state transition is expected to complete and when the software should determine the TPM has failed. These timings are called timeout. The timeouts have been broken into four categories, each with a label. The amount of time for each timeout is obtained by calling the TPM_GetCapability command. To provide an initial value (e.g., for calling the TPM_GetCapability command) for general expected behavior, a maximum for all commands that do not cause a key generation is also specified.

End of informative comment

1. There are four timeout values designated: TIMEOUT_A, TIMEOUT_B, TIMEOUT_C, and TIMEOUT_D. The time associated with each is obtained by calling the TPM_GetCapability command with the capability: TPM_CAP_PROP_TIS_TIMEOUTS. This capability SHALL return an array of UINT32 values each representing the number of microseconds for the associated timeout.
2. Until the software calls the TPM_GetCapability:TPM_CAP_PROP_TIS_TIMEOUTS, it SHOULD consider each of the timeout values as indicated in the "Default Timeouts" column in Table 14.

3. The array of timeouts SHALL be as shown in Table 14.

Table 14: Definition of Timeouts

Offset	TIMEOUT Label	Default Timeouts
[0]	TIMEOUT_A	750 millisecond
[1]	TIMEOUT_B	2 seconds
[2]	TIMEOUT_C	750 millisecond
[3]	TIMEOUT_D	750 millisecond

11.3.6 TPM_Init

Start of informative comment

The command TPM_Init is not an actual command with a defined ordinal and set of parameters; rather, it is an indication to the TPM that the Static RTM is being reset and that the RTR and RTS should also be reset. On a PC Client, this is performed using a hardware-based signal.[3] For TPM implementation using to TPM Packaging specified in Section 14, this would mean asserting the LRESET# pin. There is no requirement to use this packaging; therefore, it is up to the TPM manufacturer to define the hardware-based signal that performs this function.

End of informative comment

1. The TPM MUST implement a hardware-based signal for TPM_Init.
2. If the TPM uses the TPM Packaging specified in Section 14, this MUST be done by asserting LRESET# pin.
3. If the TPM does not use the TPM Packaging specified in Section 14, the TPM Manufacturer MUST define the pin used for TPM_Init.

11.3.7 Selftest

Start of informative comment

During the time-sensitive phase of a PC Client's startup procedure, only a small subset of the available commands is likely to be necessary. Therefore, this specification requires the TPM to perform any necessary selftest on these commands required to make them available upon completion of TPM_Init.

The Design Principles documents require each platform specific specification state the maximum time a TPM can take to continue its self-test time. Due to variations in security requirements and implementations of TPM it is difficult to mandate this to the satisfaction of all TPMs for all PC Client implementation. However, the generally accepted constraints of this platform's architecture and uses target the 1 to 2 second timeframe. PC Client platform manufacturers are advised to keep this particular aspect of the TPM's specification in mind when selecting a TPM for applicability to the platform's targeted use.

[3] This distinction is made because it is conceivable that other architectures might use other methods for performing this function.

As stated above, only a selftest for the minimum set of commands required to initialize the platform is complete after TPM_Init. The remaining selftests are required to be completed before commands that use other features are executed. This is done using either and *implicit selftest* or the TPM_ContinueSeftTest. Upon receipt of a TPM_ContinueSeftTest command, the TPM will complete all remaining and required selftests before allowing any new command. However, the TPM is also allowed to initiate a selftest upon receipt of any command that uses a TPM resource that has not been tested.

End of informative comment

1. During TPM_Init, a TPM MUST selftest all internal functions that are necessary to perform the following operations:
    a. TPM_SHA1Start
    b. TPM_SHA1Update
    c. TPM_SHA1Complete
    d. TPM_SHA1CompleteExtend
    e. TPM_Extend
    f. TPM_Startup
    g. TPM_ContinueSelfTest
2. A call TPM_GetCapability with the property TPM_CAP_PROP_TIS_TIMEOUTS MUST NOT cause the TPM to begin a selftest.
3. The maximum time to continue TPM self-test after receipt of TPM_ContinueSelfTest SHOULD be less than 1 second.

11.3.8 Input Buffer Size

Start of informative comment

Software must be aware of the maximum amount of data it can transfer to the TPM in one command. This is mostly for the NV Storage functions where large amounts of storage area can be defined by the software. This does not prevent larger areas to be defined. It means, however, that if an area is defined that requires more than the input buffer size allowed to be transferred in one command (i.e., as specified in this section), the software must break up the write into smaller pieces. This is possible because the NV Write commands allow for an offset.

Note that there is no specified output buffer size. This is because the TPM will return an error on the command before exceeding its output buffer size.

The input buffer size specified below was arrived at by calculating a reasonably large command contained within a transport session.

This size is mandated as a minimum the TPM must provide to allow software to be written to a common environment. TPMs are allowed to provide larger input buffer size to increase performance. Software that would like to take advantage of this must call TPM_GetCapability with the capability TPM_CAP_PROP_BUFFER_SIZE to get the TPM's actual input buffer size.

End of informative comment

1. The TPM MUST support an input buffer size of at least 768 bytes.

11.3.9 Errors

Start of informative comment

In general, there are four types of errors for the TPM.

1. Errors that it detects and understands and force it into an Error Mode:
    a. In this mode, the TPM responds correctly to all register reads or writes.
    b. The TPM must provide a TPM-defined Response to security operations.
    c. The TPM must allow certain TPM-defined transactions to return a response that indicates the particular error, or provides other TPM status information.
2. Errors that seem to be attacks:
    a. The TPM must completely stop responding because of these events.
        i. The TPM must not respond to reads or writes of the TPM_STS_x, TPM_ACCESS_x or any other register.
        ii. The TPM must not provide back any response.
        iii. All accesses to the TPM in this state result in an Abort until the TPM has been reset.
3. Transmission or protocol errors:
    a. STS bits and a software behavior have been defined to both identify and correct overruns or underruns on both reads and writes.
4. Errors that the TPM does not detect.

For case 1, there is no need for a status bit or interrupt, since the Response contains all the information that software needs to understand the TPM state. Case 1 may include things such as the RNG self-test failed.

For case 2, there is no need for a status bit or interrupt since the TPM does not respond to any cycles at all. Reading FFh from TPM_ACCESS_x indicates this state.

Case 3 has defined the needed status bits to detect overruns and underruns, and has provided a mechanism to recover from those errors if they are transient. Software must time out after some number of retries.

Case 4 obviously needs no status bit or other indication. Note that software may be able to determine that the TPM is in a hung or error state, even though the TPM cannot. For instance, if the TPM hung in a microcode loop, then status bits would never be updated. Software could detect this case if it has sent a command and TPM_STS_x.dataAvail never went to a “1” for minutes.

The requirement for all errors is that system security or secrets cannot be compromised.

Therefore, it is suggested to lump all errors relating to the TPM into one of the first three categories. For instance, assume there is a long power glitch. The TPM can treat this like an attack and simply cease operation, or it could go to the Error Mode and return error responses to all commands. The exact condition that forces the TPM into one error state or the other is vendor specific.

This specification defines the TPM's behavior for protocol or transmission errors. It is beyond the scope of this specification to define every hardware or software attack. It is within the scope of this specification to define the protocol for dealing with the errors, thus the above proposal.

As long as the TPM is in one of the four states above, or in the working state, it meets the requirements of this specification. The TPM must not have any point where it allows secrets or a good response to be returned when it knows there has been an error. It may use any of the above sequences to enforce this rule.

Since the transmission errors are taken care of by the protocol, the TPM has only two options for errors: go into Error Mode or to shutdown. Since each vendor's TPM will have different physical implementations, there is no good way to precisely define each error and whether it should go into Error Mode or do a shutdown. Therefore, it is vendor specific whether a particular error causes shutdown or Error Mode responses.

Software has two cases to deal with. If the TPM has shutdown, this is detected by TPM_STS_x.commandReady or TPM_STS_x.dataAvail never being asserted, and the protocol is defined to handle that with appropriate timeouts. If the TPM returns Error Responses, then the software is already designed to handle this case.

Since the recovery for shutdown is system reset and the recovery for Error Mode is also system reset, there is no need to define in more details how the TPM should handle each error. In the future, if an error has a more graceful recovery mechanism, then the specification will need to be more precise on how the TPM must handle the error. Today, the software stack will simply detect timeout in the protocol and reset or it will detect that the TPM is only returning Error Responses for all TPM commands and reset.

End of informative comment

The requirement for all errors is that system security or secrets MUST NOT be compromised.

As long as the TPM is in one of the four states above, or in the working state, it meets the requirements of this specification. The TPM MUST NOT have any point where it allows secrets or a good response to be returned when it knows there has been an error. It MAY use any of the above sequences to enforce this rule.

For the transmission or protocol error case, software MUST time out after some number of retries.

Since the transmission errors are taken care of by the protocol, the TPM has only two options for errors: go into Error Mode or to shutdown. Since each vendor's TPM will have different physical implementations, there is no good way to precisely define each error and whether it SHOULD go into Error Mode or do a shutdown. The TPM MAY either shutdown or go into Failure Mode for any particular error.

11.3.10 Access Register

Start of informative comment

The purpose of this register is to allow the processes operating at the various localities to share the TPM. The basic notion is that any locality can request access to the TPM by setting the TPM_ACCESS_x.requestUse bit using its assigned TPM_ACCESS_x register address. If there is no currently set locality, the TPM sets current locality to the requesting one and allows operations only from that locality. If the TPM is currently at another locality, the TPM keeps the request pending until the currently executing locality frees the TPM. Software relinquishes the TPM's locality by writing a "1" to the TPM_ACCESS_x.activeLocality bit. Upon release, the TPM honors the highest locality request pending. If there is no pending request, the TPM enters the "no locality" phase.

There may be circumstances where the access to the TPM is "held" by either crashed or ill-behaved software. For this reason, the TPM_ACCESS_x.Seize bit may be used. It is generally assumed that software executing at higher level localities is more trusted and less prone to crashing and better behaved at relinquishing the TPM. The TPM_ACCESS_x.Seize bit allows higher-level localities to gain control of the TPM. This method, however, should be the exception rather than the common method for gaining access to the TPM.

End of informative comment

Except when writing to the TPM_ACCESS.x. Seize bit, all writes to this register MUST contain only one bit set to a "1". If the TPM receives a write with more than 1 bit set, the TPM MUST ignore the entire cycle. For each write, bits containing a "0" are ignored.

Table 15: Access Register

Abbreviation:			TPM_ACCESS_x	
General Description:			Used to gain ownership of the TPM	
Bit Descriptions:				
7	Read Only	tpmRegValidSts	Default: 0,	This bit is a "1" to indicate that the other bits in this register are valid. If this bit is "0", then software SHOULD ignore the other bits in this register. The main usage of this bit is to indicate when bit [0] is valid. Before bit [7] is set, the TPM may not have had time to read non-volatile storage and place the correct value into bit [0]. If this bit is never set, tsoftware MUST assume that the TPM is broken and not attempt to use it. This bit is "0" at reset and remains a "0" until the TPM has gone through its self test and initialization and has established correct values in the other bits.
6	Read Only	Reserved	Default: 0	MUST return "0"
5	Read/ Write	activeLocality	Default: 0	0 = This locality is not active. 1 = This locality is active. Write a "1" to this bit to relinquish control of this locality. Writing a "1" to this bit MUST clear both the TPM_ACCESS_x.activeLocality bit and the requestUse bit. If the requestUse bit is set and software decides it does not need to use the TPM, it writes a "1" to the TPM_ACCESS_x.activeLocality bit to clear the request. If the TPM has moved to make this locality active, writing a "1" to this bit would also clear the TPM_ACCESS_x.activeLocality setting. When there are multiple pendingRequest bits set, and the TPM is arbitrating for the next user, it MUST choose the highest locality that has its pendingRequest bit set. Timeouts: 1. If there is no locality set, this bit MUST be valid within the time specified by TIMEOUT_A. 2. If another locality is active the time for this locality's bit to be active (i.e., this locality obtains control of the TPM) will depend on the external operating environment and is non-deterministic from the TPM's perspective.
4	Read/ Write	beenSeized	Default: 0	Set to "1" to indicate that this locality had the TPM taken away while this locality had the TPM_ACCESS_x.activeLocality bit set. Software can use this bit to determine if it needs to abort an entire task and begin it again when it gets the TPM back again. Writing a "1" to this bit clears it.

Abbreviation:				TPM_ACCESS_x
General Description:				Used to gain ownership of the TPM
Bit Descriptions:				
3	Write Only	Seize	Reads always return 0	If the locality setting this bit to a "1" is equal to or less than the current locality, the TPM MUST ignore this command. The rest of the statements in the description of this bit apply if, and only if, the locality setting this bit to a "1" is greater than the current locality. 1. When software writes a "1" to this bit the TPM MUST reset the TPM_ACCESS_x.activeLocality bit and remove ownership for localities less than the locality which is writing this bit. For example, writing a "1" to this bit using Locality 1 MUST force the TPM to reset the TPM_ACCESS_x.activeLocality bit for Locality 0 if it is set. The TPM MUST stop executing commands from the Locality 0 port. Likewise, if Locality 2 were to write a "1" to this bit, then the TPM would reset TPM_ACCESS_x.activeLocality for either Locality 0 or Locality 1, whichever was active. For all descriptions for the setting of the Seize bit, the description applies only if successful per the above description. 2. Setting this bit does not affect the state of the TPM_ACCESS_x.requestUse bit for any locality except the one issuing the Seize bit. For the locality issuing the Seize bit, the TPM MUST set the TPM_ACCESS_x.activeLocality bit and clear the TPM_ACCESS_x.requestUse bit where x is the locality that issued the Seize. 3. The TPM MUST ignore any write to this bit from Locality 0 since this operation has no real meaning for Locality 0, since it is the lowest on the locality pole except for the legacy port. 4. When a write to TPM_ACCESS_x occurs, and bit [3] (Seize) is set, then bits [5] and [1] are "don't cares" if they were set in the same write. If either of these bits is set, the cycle is treated as a write only to Seize and the other bits are ignored. Writing bit[4] at the same time as Seize MUST clear the TPM_ACCESS_x.beenSeized bit and do a regular Seize. 5. Writing a "1" to this bit MAY also abort a command that is in process. Whether the abort happens or not is TPM specific and software MUST NOT assume that the abort was done. 6. After a successful Seize operation, the software should poll activeLocality bit to determine if Seize operation was successful. Software then reads the TPM_STS_x.commandReady bit. a. If the TPM_STS_x.commandReady bit is a "0", software SHOULD write a "1" to the bit and then poll until it becomes a "1". b. If the TPM_STS_x.commandReady bit is set, software MAY immediately write the command to the TPM.
2	Read Only	pendingRequest	Default: 0	1 – some other locality is requesting usage of the TPM 0 – no other locality is requesting use of the TPM This bit can be used by software to determine if it should relinquish control of the TPM so some other locality can use it.
1	Read/ Write	requestUse	Default: 0	0 – no pending request to use this locality 1 – this locality is requesting to use TPM This bit is cleared to "0" by the TPM when it selects this locality to become the next user of the TPM. It is also cleared if there is a write of "1" to TPM_ACCESS_x.activeLocality. It is not cleared by writing a "0" or a "1" to TPM_ACCESS_x.requestUse.

Abbreviation:	TPM_ACCESS_x			
General Description:	Used to gain ownership of the TPM			
Bit Descriptions:				
0	Read Only	tpmEstablishment	Default: 1	There are some special end cases (e.g., error conditions) where software needs to know if a T/OS has even been established on this platform. This bit performs this function. Meaning of this bit: 1. A value of "1" indicates that a T/OS has not been established on the platform and that no secrets that are to be protected by a T/OS have existed in memory. 2. A value of "0" indicates that either currently or at some point in the past, a T/OS has been established; therefore, either currently or at some point in the past, the contents of memory may have contained secrets that were protected by the T/OS. 3. This bit MUST return a value of "1" until the first time that a secure environment has been established. 4. Upon receipt of the TPM_HASH_START LPC command, the TPM MUST clear this bit to a value of "0". 5. This bit MUST be kept in NV STORAGE and is persistent across all TPM resets, power cycles, initialization, and ownership changes. 6. This bit MUST be updated before TPM_STS_x.commandReady is asserted for the next command. 7. This bit MUST be set to "1" upon receipt of a valid TSC_ResetEstablishmentBit as defined in the TPM Main Specification. *Application Note*: This bit MAY be a "0" at reset and set to a "1" (assuming that no TPM_HASH_START command has ever been received) after TPM internal initialization is complete. Bit[7] MUST NOT be set until this bit has the correct value. Once RESET# is de-asserted, this bit MUST NOT return a "1", even if bit[7] is marked as invalid, unless "1" is the known correct value. Note that this bit is cleared by the TPM_HASH_START commands. The TPM_HASH_* LPC commands do not depend on TPM_STS_x.commandReady, but the TPM MUST NOT assert TPM_STS_x.commandReady for the next command until the TPM_ACCESS_x.tpmEstablishment bit is cleared following these commands.

11.3.11 Status Register

Start of informative comment

Use of the write to the TPM_STS_x commandReady bit is overloaded. If there is no command being executed, a write to this bit is an indicator to the TPM that it must prepare to be ready to receive a command. If there is a command being executed, this bit serves as an abort of that command. If a command has completed and the results have been read, a write to this bit allows the TPM to free internal resources (including the Read and Write FIFOs) and proceed with background or other processes allowed during idle time. A TPM may be designed in a manner that allows the first write to this bit to clear and free the TPM's resources and make it ready to receive a command.

Thus, the software MUST be prepared to send two writes of a "1" to this bit: the first to indicate successful read of all the data, thus clearing the data from the ReadFIFO and freeing the TPM's resources, and the second to indicate to the TPM it is about the send a new command. The time between receiving the data from a command and sending the first write to this bit should be very short to allow TPMs that perform background processing to proceed. The time between the first write and the second that indicates the beginning of a new command is arbitrary.

The software may be written such that the second write to this bit is only necessary if the TPM does not respond with a ready after the first write. In this case, the software, after writing a "1" to this bit indicating the receipt of the data, may query this bit. If the TPM sets this bit to a “1” indicating its readiness to receive a command, the software may proceed to send the command without writing a “1” to this bit.

End of informative comment

For each write to this register, there MUST be only 1 bit set to a “1”. If the TPM receives a write with more than one bit set, the TPM MUST ignore the entire cycle. For each write, bits containing “0” are ignored.

The TPM is considered to be in the following defined states:

1. *Command Reception*: is between the write of the first byte of a command to the WriteFIFO following a ready state and the receipt of the TPM_STS_x.tpmGo.
2. *Command Execution*: is after receipt of TPM_STS_x.tpmGo and the TPM setting TPM_STS_x.commandReady:dataAvail to a “1” unless the command is aborted.
3. *Command Completion*: is after completion of a command (indicated by the TPM setting the TPM_STS_x.commandReady:dataAvail to a “1” and before a write of a “1” by the software to TPM_STS_x.commandReady:commandReady).
4. *Idle*: is any time after Command Completion followed by the write of a “1” by the software to TPM_STS_x.commandReady, following locality change, or an abort. Idle is the initial state of TPM upon completion of TPM_Init.
5. *Ready*: is any time the TPM is ready to receive a command.

The following informative diagram is derived from the above normative statements. It is informative and only for illustrating diagrammatically the above TPM states and their transitions. The numbers in parentheses reference the states represented the by row number in Table 19 on page 57.

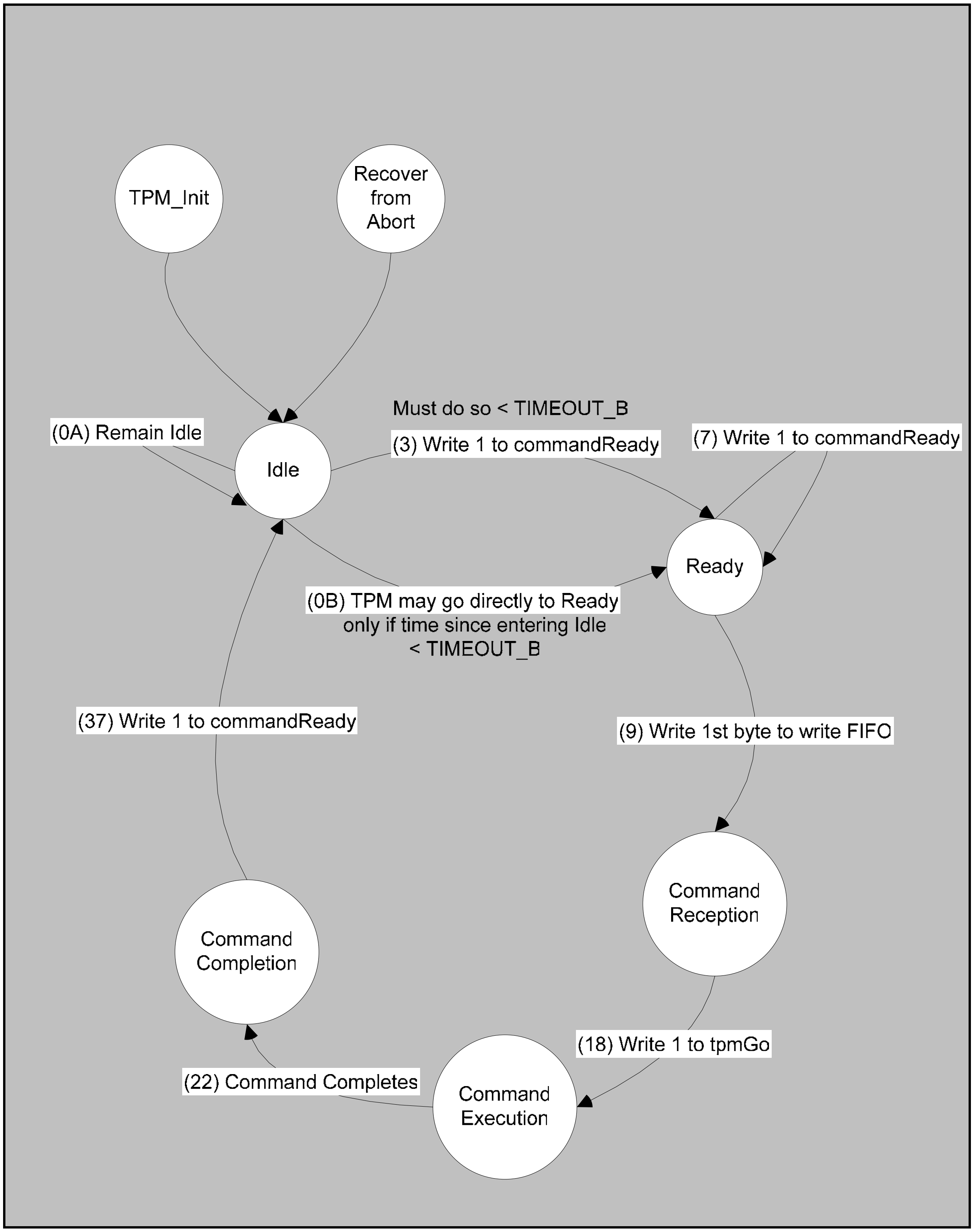
TPM_Init
Recover from Abort
Must do so < TIMEOUT_B
(0A) Remain Idle
Idle
(3) Write 1 to commandReady
(7) Write 1 to commandReady
Ready
(0B) TPM may go directly to Ready only if time since entering Idle < TIMEOUT_B
(37) Write 1 to commandReady
(9) Write 1st byte to write FIFO
Command Reception
Command Completion
(18) Write 1 to tpmGo
(22) Command Completes
Command Execution

Figure 1: State Transition Diagram

Table 16 describes the bits of the status register.

Table 16: Status Register

Abbreviation:	TPM_STS_x			
General Description:	Contains general status details			
Bit Descriptions:				
23 :8	Read Only	burstCount	Default = number of consecutive writes that can be done to the TPM	Indicates the number of bytes that the TPM can return on reads or accept on writes without inserting LPC long wait states on the LPC bus. This field MAY be dynamic, where it changes with each byte that is read or written. The field MAY also be static, where it reports some fixed number and reports “0” anytime that fixed number of bytes is not available. Software SHOULD read TPM_STS_x.burstCount and then read or write that number of bytes before reading TPM_STS_x.burstCount again. In the static case, software could read TPM_STS_x.burstCount and get some value, say 16, and then do one read, and re-read TPM_STS_x.burstCount and get a value of “0”, even though it had only read 1 byte. If software polls on every read, then it MUST wait until the TPM has processed the first byte and has set TPM_STS_x.burstCount to some non-zero value again. NOTE: For any design that is not a true counter, the TPM MUST NOT be designed with the dependency that it does not internally process reads or writes until it has received the number given in the TPM_STS_x.burstCount. In the case above, if software were to read “0” after the first byte had been moved, and wait for TPM_STS_x.burstCount to go non-zero, while the TPM is waiting for 15 more reads before setting the TPM_STS_x.burstCount back to 16, then the software deadlocks. If the TPM has a 16-deep ReadFIFO and gets one read and, therefore, sets the COUNT to “0”, then it MUST start moving more data into the ReadFIFO when the first byte has been read. For WriteFIFO’s, the TPM MUST process the bytes as they come in and not wait until the entire WriteFIFO has been filled. In the above case, if the TPM reports a truly dynamic TPM_STS_x.burstCount, then there is no issue, since the TPM_STS_x.burstCount only goes to “0” when the FIFO is full for writes, or empty for reads. Therefore, the TPM could delay loading more bytes into the ReadFIFO or removing bytes from the WriteFIFO until they are empty/full, and this will not create the deadlock that arises from reporting “0” in TPM_STS_x.burstCount when the FIFO is not truly empty/full. The hardware MUST be designed to report the correct count even though there is a time delay in returning the 2 bytes on the LPC bus. A couple of solutions: 1. Return 0x00 for the upper byte in all cases. This limits the field to 255 bytes, but that is a sufficiently large number that polling on this register every 255 bytes is insignificant in terms of performance. 2. Guarantee that the count does not change between the read of the first byte and the read of the second byte. a. This could be done if the hardware updates the count field at the time of the previous read or write and does not change it until the next read or write. b. It could be done if hardware latches both bytes of the count that is returned on the read of the first byte, and returns the latched second byte on the next read. In this case, if there is a read or write of the data before the next read of the TPM_STS_x.burstCount, then the latch is reset. c. The field is used statically. d. Note that there could be the case where internally the TPM is processing the FIFO and TPM_STS_x.burstCount is dynamic, whereby the count is changing even though there are no LPC bus transactions. In this case, the read of the low byte might not be synchronized with the high byte – hardware must not allow this condition. If TPM_STS_x.dataAvail is set, then this register provides the number of reads that can be done without LPC long wait states. If TPM_STS_x.dataAvail is not set, then this register provides the number of writes that can be done without LPC long wait states. Software SHOULD read this field and read/write the number of bytes indicated. Then it SHOULD read the field again to see if it can read/write more data. Software SHOULD NOT read the field without sending/receiving the number of bytes indicated by the previous read. If it does, it MAY read “0” or a value less than the previous value minus the number of transactions done. In the case where the total command or response has fewer bytes than the TPM_STS_x.burstCount field indicates, software MUST NOT pad the reads or writes to the count value. Once the command is written, the TPM_STS_x.tpmGo bit MUST be written and

Abbreviation:	TPM_STS_x			
General Description:	Contains general status details			
Bit Descriptions:				
				that clears the count field for the response data. Timeout: TPM_STS_x.burstCount MUST be valid within the time specified by TIMEOUT_D after TPM_STS_x.stsValid is valid. NOTE*:* It takes roughly 330 ns per byte transfer on LPC. 256 bytes would take 84 μs, which is a long time to stall the CPU. Chipsets may not be designed to post this much data to LPC; therefore, the CPU itself is stalled for much of this time. Sending 1 kB would take 350 μs. Therefore, even if the TPM_STS_x.burstCount field is a high value, software SHOULD be interruptible during this period.
7	Read Only	stsValid	Default: 0	This bit indicates that both TPM_STS_x.dataAvail and TPM_STS_x.Expect are correct. If TPM_STS_x.stsValid is not set, then TPM_STS_x.dataAvail and TPM_STS_x.Expect are not guaranteed to be correct and software that is using TPM_STS_x.dataAvail or TPM_STS_x.Expect must poll on TPM_STS_x register until TPM_STS_x.stsValid is set. The TPM MUST set the TPM_STS_x.stsValid bit within TIMEOUT_C after the last data cycle is received.
6	Read/ Write	commandReady	Default: 0	**Read**: 1. 1 indicates that the TPM is in the *Ready* state indicating it is ready to receive a new command. a. The TPM MUST not enter the *Ready* state unless both the ReadFIFO and WriteFIFO are empty. b. Software MUST start sending a command to the TPM only when the TPM is in this state. c. The TPM MUST clear this bit to 0 when the first byte of data is received into the WriteFIFO. 2. 0 indicates the TPM is not in a *Ready* state indicating it is not ready to receive a new command. 3. The TPM MUST clear this bit to 0 when the first byte of data is received into the WriteFIFO. A write of "0" to this bit MUST be ignored. Upon a write of a "1" to this bit: 1. A write of "0" is illegal and MUST be ignored. 2. When in the *Ready* state: a. The TPM MUST ignore this write and remain in the *Ready* state. 3. When in the *Idle* state: a. The TPM MUST enter the *Ready* state within the time TIMEOUT_B. 4. When in the *Command Reception* state: a. The TPM MUST treat this as an abort according to Section 11.3.3. 5. When in the *Command Completion* state: a. The TPM MUST clear the ReadFIFO and the WriteFIFO. b. The TPM MUST enter either the *Idle* state or the *Ready* state. c. If the TPM does not enter the *Ready* state within time TIMEOUT_B, it MUST enter the *Idle* state. 6. When in the *Command Execution* state: a. The TPM MUST cause the currently executing command to be aborted according to Section 11.3.3.
5	Write Only	tpmGo	Reads always return 0	After software has written a command to the TPM and sees that it was received correctly, software MUST write a "1" to this bit to cause the TPM to execute that command. Note that once this bit is written, the TPM MUST execute the received command. That may take many seconds to minutes for certain commands, such as key generation.
4	Read Only	dataAvail	Default: 0	This bit indicates that the TPM has data available as a response. When set to "1", software MAY read the ReadFIFO. The TPM MUST clear the bit to "0" when it has returned all the data for the response. The TPM sets this bit when it is ready to return the response. Software MUST NOT read the

Abbreviation:	TPM_STS_x			
General Description:	Contains general status details			
Bit Descriptions:				
				ReadFIFO until TPM_STS_x.dataAvail is set. The TPM MUST clear this bit to "0" when it has returned the last byte of the response. Note that there may be some time delay between the read of the last byte from the ReadFIFO and when TPM_STS_x.dataAvail is cleared. The TPM MUST long wait state the LPC bus on reads to TPM_STS_x until it can guarantee that TPM_STS_x.dataAvail returns the correct value. The TPM MUST NOT report that there is more data to return, when in reality it has returned the last byte. To detect overruns/underruns, software SHOULD read TPM_STS_x.dataAvail before it reads what it thinks is the last byte of the response. If TPM_STS_x.dataAvail is "1", then there is at least 1 more byte to read. Software reads the last byte and re-reads TPM_STS_x.dataAvail. In this case, TPM_STS_x.dataAvail should be "0"; since if things are working correctly, there is no more data to return. If TPM_STS_x.dataAvail is still "1", then the TPM has more data to return, and software is out of sync with the hardware; therefore, software should write TPM_STS_x.responseRetry to force the TPM to resend the response. If the read of TPM_STS_x.dataAvail before software reads the last byte returns a "0", the TPM thinks it has no more data to return, while software still expects 1 more byte. In this case, software MUST write TPM_STS_x.responseRetry to force the TPM to resend the response. When the response has been correctly received, software SHOULD write to TPM_STS_x.commandReady to indicate the response was correctly received, the ReadFIFO can be cleared, and the WriteFIFO is enabled for the next command. This bit clears to "0" when the TPM has sent the last byte of the response. It remains a "0" even if there are more reads to the now-empty ReadFIFO. If software writes to TPM_STS_x.responseRetry, then the TPM MUST set TPM_STS_x.dataAvail back to "1". Software MUST wait until TPM_STS_x.dataAvail has been set to "1" before reading the ReadFIFO. TPM_STS_x.dataAvail is cleared if software writes a "1" to TPM_STS_x.commandReady even though the response data had not been read.
3	Read Only	Expect	Default: 0	The TPM sets this bit to a value of "1" when it expects another byte of data for a command. It clears this bit to a value of "0" when it has received all the data it expects for that command, based on the TPM size field within the packet. This bit is set either after TPM_STS_x.commandReady is written and the WriteFIFO is ready to receive data (TPM is entering the ready state), or after the first byte of the command has been written into the TPM's Receive FIFO (TPM is entering the command execution state). This bit MUST stay set until the TPM has received the number of bytes it expects for that command. Software can check for overruns and underruns as follows: 1. Software writes all the bytes of the command except for the last byte. It reads TPM_STS_x.Expect. If TPM_STS_x.Expect is "1" then there have been no errors so far. If it is a "0" then there was an error, and software should restart the command. 2. If software read a"1" above, then it writes the last byte and re-reads TPM_STS_x.Expect. If TPM_STS_x.Expect is a "0", the command was successfully received; if it is a "1", then there was a failure and the command should be re-issued.
2	Read Only	Reserved (0)	Reads always return 0	
1	Write Only	responseRetry	Reads always return 0	Software writes a "1" to this bit to force the TPM to re-send the response. Reads MUST return "0"
0	Read Only	Reserved (0)	Reads always return 0	

11.3.12 Data FIFO Register

Table 17: Data FIFO Register

Abbreviation:	TPM_DATA_FIFO_x
General Description:	Data port for TPM

Bit Descriptions:			
7:0	Data	Default: undefined	Software reads packet return data and return status from this port. The return packet for a command is multiple bytes, but software reads it 1 byte at a time. Software SHOULD read the TPM_STS_x.burstCount field to determine how many consecutive bytes it may read. A read to the ReadFIFO when the TPM_STS_x.stsValid bit indicates there is no data and SHOULD return FFh. Software writes packets to this port. The packet for a command is multiple bytes, but software writes it 1 byte at a time. Software SHOULD read the TPM_STS_x.burstCount field to determine how many consecutive bytes it may send to the TPM. The TPM MUST not drop a write on the LPC bus when it is not able to accept it. Instead, it MUST wait-state the LPC bus. The process calling TPM_HASH_* LPC commands function MUST not poll the count field and MUST send its data to the TPM using only the LPC protocols and bus speeds. Software SHOULD read the TPM_STS_x.burstCount field for better general system performance.

11.4 Interface Capability

Table 18: Interface Capability

Abbreviation:				TPM_INTF_CAPABILITY_x
General Description:				Provides information of which interrupts that particular TPM supports. The TPM MUST implement this register.
Bit Descriptions:				
31:9	Read Only	Reserved	Reads always return 0	
8	Read only	BurstCountStatic	Default: Defined by hardware	Indicates whether the TPM_STS_x.burstCount field is dynamic or static 1 = TPM_STS_x.burstCount is static 0 = TPM_STS_x.burstCount is dynamic
7	Read only	CommandReadyIntSupport	Default: Defined by hardware	Corresponds to TPM_INT_ENABLE_x.commandReadyEnable 1 = supported 0 = not supported
6	Read Only	InterruptEdgeFalling	Default: Defined by hardware	Falling edge interrupt support 1 = supported 0 = not supported
5	Read Only	InterruptEdgeRising	Default: Defined by hardware	Rising edge interrupt support 1 = supported 0 = not supported
4	Read Only	InterruptLevelLow	Default: Defined by hardware	Low level interrupt support. This interrupt trigger is mandatory. 1 = supported 0 = illegal:
3	Read Only	InterruptLevelHigh	Default: Defined by hardware	High level interrupt support 1 = supported 0 = not supported
2	Read Only	LocalityChangeIntSupport	Mandatory: MUST be 1	Corresponds to TPM_INT_ENABLE_x.localityChangeIntEnable. This is a mandatory interrupt. 1 = supported 0 = not allowed
1	Read Only	stsValidIntSupport	Default: Defined by hardware	Corresponds to TPM_INT_ENABLE_x.stsValidIntEnable 1 = supported 0 = not supported
0	Read Only	dataAvailIntSupport	Mandatory: MUST be 1	Corresponds to TPM_INT_ENABLE_x.dataAvailIntEnable. This is a mandatory interrupt. 1 = supported 0 = not allowed

11.5 Status Bit State Transitions

Table 19 shows the changes in status bits based on the command or action done to the TPM. Notice, this is not a state transition table covering the states defined in Section 11.3.11 rather a table describing how the status bits change based on initial condition and action taken. The following rules apply to Table 19.

1. There MAY be intermediate status bit states, where a command has finished, but TPM_STS_x.dataAvail is not yet set. Software is expected to poll until the appropriate status bit is set.
2. The bits in the table represent values only when TPM_STS_x.stsValid = 1. Transitional states when TPM_STS_x.stsValid = 0 are neither captured nor represented in this table.
3. This table applies only to status bit states where a locality has already been selected and no change in locality is performed.
4. The statements in the column labeled “Action Taken” are informative when shaded and are derived from normative statements contained within the definitions of the TPM_STS_x register in Section 11.3.11. If there is an inconsistency between this column and the statements within normative definitions of the TPM_STS.x registers, the normative statements within definitions of the TPM_STS.x registers take precedence. The statements made in unshaded text and delimited by the phrases: “Start of normative comment” and “End of normative comment” are normative. Note, this is a reversal of the standard notation.
5. Normal transitions are highlighted in yellow and are indexed to the state transition illustrated in Figure 1.
6. In all cases in Table 19 where Idle is the next state, the TPM is allowed to transition directly to Ready effectively via transition 0B. This simplifies the table by not having to show two ending states possibilities. When makinga transparent transition to the Ready state, the TPM is not required to indicate it is or has been in the Idle state to the software, therefore, making this transition transparent to the software.
7. The following abbreviations are used in Table 19:

Label	Bit Definition
C/R	TPM_STS_x.commandReady
D/A	TPM_STS_x.dataAvail
E	TPM_STS_x.Expect
TG	TPM_STS_x.tpmGo
R/R	TPM_STS_x.responseRetry
N/C	No Change to this bit from previous state
—	Nothing written to the TPM for this bit
X	Either 0 or 1. The TPM is allowed to maintain this bit as either value for this state. Software MUST be capable of managing the TPM if either case is implemented.

Table 19: State Transition Table

#	Present State of TPM_STS_x					Bits / Data Written to TPM_STS_x or TPM_DATA_FIFO.x						Next State & Result / Reason				Action Taken
	TPM State	C/R	D/A	E		C/R	TG	R/R	Write Data	Read Data		TPM State	C/R	D/A	E	*Informative*
0A	Idle	0	0	0		—	—	—	—	—		Idle	0	0	0	***Start of normative comment*** I_T = Time since entering the Idle state If TPM is Idle and I_T >= TIMEOUT_B Then: TPM MUST remain in the Idle state (i.e., continue executing in this in this row/state). until commanded to change by the software (i.e., via state transition in row 3). Else: (i.e., I_T < TIMEOUT_B) TPM MAY transition to the Ready state (i.e., transition to row 0B) ***End of normative comment***
0B	Idle	0	0	0		—	—	—	—	—		Ready	1	0	X	This specification allows a TPM to be implemented such that the software never sees the Idle state. This transition codifies and enables that behavior. ***Start of normative comment*** I_T = Time since entering the Idle state If TPM is Idle and I_T >= TIMEOUT_B Then: TPM MUST NOT enter the Ready state (i.e., it MUST NOT execute the state transition in this row and MUST remain Idle in Row 0A). Else: (i.e., I_T < TIMEOUT_B) TPM MAY transition to the Ready state (i.e., execute the transition in this row). If the TPM performs this transition, it is not required to indicate that it is, or has been, in the Idle state; rather it is allowed to appear as if it went directly from the state prior to the Idle state to the Ready state transparently to the software. ***End of normative comment***
1	Idle	0	0	0		0	0	1				Idle	0	0	0	There is no response to retry. No state change.
2	Idle	0	0	0		0	1	0				Idle	0	0	0	There is no command to execute. No state change.
3	Idle	0	0	0		1	0	0				Ready	1	0	X	This is the typical state change resulting from the software's request to send a command.
4	Idle	0	0	0					Write data			Idle	0	0	0	TPM drops write, since it is not expecting data. If the software was sending a command and still has data to write, then this is an error in transmission and the software MUST re-transmit the command.

#	Present State of TPM_STS_x					Bits / Data Written to TPM_STS_x or TPM_DATA_FIFO.x						Next State & Result / Reason				Action Taken
	TPM State	C/R	D/A	E		C/R	TG	R/R	Write Data	Read Data		TPM State	C/R	D/A	E	***Informative***
5	Idle	0	0	0						Read data		Idle	0	0	0	TPM returns FFh, since TPM_STS_x.dataAvail is not set.
6	Ready	1	0	X		0	0	1				Ready	1	0	N/C	No effect on TPM, since TPM_STS_x.commandReady indicates that the response has been read successfully.
7	Ready	1	0	X		0	1	0				Ready	1	0	N/C	Causes no change to the TPM, since there is no command to process.
8	Ready	1	0	X		1	0	0				Ready	1	0	N/C	No effect on TPM, since it is ready to receive commands.
9	Ready	1	0	X					Send first byte			Reception	0	0	1	Clear TPM_STS_x.commandReady on first byte.
10	Ready	1	0	X						Read Data		Ready	1	0	N/C	No effect on TPM. TPM returns FFh.
11	Reception	0	0	1		0	0	1				Reception	0	0	1	There is no response to retry. No state change.
12	Reception	0	0	1		0	1	0				Reception	0	0	1	TpmGo is not valid at this time. TPM ignores this state transition.
13	Reception	0	0	1		1	0	0				Idle	0	0	0	The command being sent to the TPM is aborted.
14	Reception	0	0	1					Send more data other than last byte			Reception	0	0	1	
15	Reception	0	0	1					Send last data			Reception	0	0	0	Good transmission if the software has only one more byte. If either software has more than one more byte to send or if expect = 1 and software has not more data to send, this is a transmission error and the software must resend the command.
16	Reception	0	0	1						Read data		Reception	0	0	1	TPM returns FFh.
17	Reception	0	0	0		0	0	1				Reception	0	0	0	Nothing to retry.
18	Reception	0	0	0		0	1	0				Execution	0	0	0	This is the normal transition from sending the command to the start of execution of the command.
19	Reception	0	0	0		1	0	0				Idle	0	0	0	Aborts the command sent.
20	Reception	0	0	0					Write			Reception	0	0	0	Write is not expected. Drop write. TPM ignores this state transition.
21	Reception	0	0	0						Read		Reception	0	0	0	Read 0xFF.
22	Execution	0	0	0		—	—	—				Completion	0	1	0	Upon command completion, TPM sets TPM_STS_x.dataAvail to a 1.
23	Execution	0	0	0		0	0	1				Execution	0	0	0	There is no response to retry. No state change.
24	Execution	0	0	0		0	1	0				Execution	0	0	0	The TPM is already executing a command. No state change.
25	Execution	0	0	0		1	0	0				Idle	0	0	0	The executing command is aborted.

#	Present State of TPM_STS_x					Bits / Data Written to TPM_STS_x or TPM_DATA_FIFO.x						Next State & Result / Reason				Action Taken
	TPM State	C/R	D/A	E		C/R	TG	R/R	Write Data	Read Data		TPM State	C/R	D/A	E	*Informative*
26	Execution	0	0	0					Write data			Execution	0	0	0	TPM drops write, since it is not expecting data. If the software was sending a command and still has data to write, then this is an error in transmission and the software MUST re-transmit the command.
27	Execution	0	0	0						Read data		Execution	0	0	0	TPM returns FFh, since TPM_STS_x.dataAvail is not set.
28	Completion	0	1	0		0	0	1				Completion	0	1	0	TPM MUST reset ReadFIFO pointers and start sending the response from the first byte.
29	Completion	0	1	0		0	1	0				Completion	0	1	0	Causes no change to the TPM, no new command to execute.
30	Completion	0	1	0		1	0	0				Idle	0	0	0	Aborts command.
31	Completion	0	1	0					Write data			Completion	0	1	0	No effect on TPM, it is not accepting a command.
32	Completion	0	1	0						Read data other than last byte		Completion	0	1	0	TPM returns data.
33	Completion	0	1	0						Read last byte		Completion	0	0	0	Good transmission, since TPM_STS_x.dataAvail = 0.
35	Completion	0	0	0		0	0	1				Completion	0	1	0	Normal start of response/Retry operation.
36	Completion	0	0	0		0	1	0				Completion	0	0	0	TpmGo is not valid at this time. TPM ignores this state transition.
37	Completion	0	0	0		1	0	0				Idle	0	0	0	This is the typical state change resulting from the software's indication that it received the results of the command, and the TPM may proceed to the Idle state and, depending on implementation, proceed directly to the Ready state.
38	Completion	0	0	0					Write			Completion	0	0	0	Write is not expected. Drop write. TPM ignores this state transition.
39	Completion	0	0	0						Read		Completion	0	0	0	Read 0Xff.
40	Any					More than 1 bit set on any status write						No change to state				Software error, TPM ignores command and does nothing

12. Interrupts

Start of informative comment

The method for asserting interrupts uses the Serial-IRQ (SIRQ) protocol for interrupts. The protocol emulates the set of individual hardware signals using time division multiplexing between frames. An understanding of the SIRQ protocol is critical to the TPM implementer. The direct assertion of the SIRQ line does not, in itself, signal an interrupt. The assertion of the interrupt is a combination of the change in the SIRQ signal during a time slot designated for that interrupt number. The state (level, edge, high, low) is expressed as the state of the SIRQ line during the assigned time slot over a series of frames. The description below the term interrupt refers to the assertion of an interrupt using the SIRQ protocol as just described.

The TPM must be designed to support asserting any of the IRQ[0:15]. Certain platforms may not support certain IRQs being assigned to the TPM; therefore, the TPM must be capable of asserting any of the 16 possible IRQs. The TPM must not assert PIRQ[A:D] in the SIRQ stream.

There is a capability register that allows each TPM to indicate which interrupts it supports.

Because use to the TPM is non-preemptive (except for the special case of Seize) and the TPM is single threaded, there is no issue with regard to sharing or colliding interrupts across localities. For example, if one locality starts a TPM operation, it cannot release the TPM to another locality until the pending TPM operation completes. However, if one locality (e.g., Locality 0) starts a long TPM operation, then turns control to another locality (e.g., Locality 2) before the long operation completes (and of course does not relinquish the TPM which would cause an abort), the second locality (e.g., Locality 2) would not know what the interrupt is for. This situation is outside the purview of this specification and negotiation of interrupt handling is to be done by the software.

The TPM reports all schemes it supports in the Interrupt Capabilities register bits 3-6. The software selects the scheme using the Interrupt Enable register bits 3-4. If the TPM supports only one scheme, bits 3 and 4 MAY be read only and return the value of the implemented scheme.

When an event occurs that causes the TPM to signal an interrupt, the TPM must set the appropriate bits in the TPM_INT_STATUS_x register. If the TPM has not already sent an interrupt, the “0” to “1” transition of a bit in the TPM_INT_STATUS_x register must cause the TPMs to assert the appropriate interrupt per the SIRQ protocol. The interrupt service routine will read the TPM_INT_STATUS_x register to determine the cause and take appropriate action. When the interrupt has been serviced, the interrupt service routine must do an End-of-Interrupt (EOI) to the I/O APIC to re-arm the TPM’s interrupt in the I/O APIC. Then the interrupt service routine must also send a TPM_EOI to the TPM to allow it to send new interrupts.

The TPM must not issue another interrupt until it has received its TPM_EOI message (see below), even if new events occur that should cause an interrupt. The TPM should set the appropriate bit in TPM_INT_STATUS_x, but the actual assertion of the interrupt will only occur after the TPM_EOI. If the interrupt handler detects multiple bits set in TPM_INT_STATUS_x, it may handle all the causes and clear multiple status bits. This means that an interrupt may be handled without actually causing a new interrupt.

The TPM_EOI to the TPM is writing a “1” to the bit in the TPM_INT_STATUS_x register that corresponds to the type of interrupt just handled. Software may write multiple bits if it has handled multiple interrupt causes at one time.

The following sections are added for clarification. They should not change functionality.

If there are multiple bits set in the Interrupt register, and software does not clear all the interrupts, then the TPM must issue another interrupt after it sees the TPM_EOI, which is a write to the Interrupt register.

The software must not change the state of the TPM_INT_ENABLE_x globalIntEnable flag while an interrupt is active.

For example: if after the write to the Interrupt register, there are bits still set, the TPM issues another interrupt. If software writes all the bits of the Interrupt register, so that after the write the register = “0”, no new interrupt would be generated.

This covers the case that software handles one interrupt at a time and then returns. It also covers the case that software handles all the interrupts it knows about, so writes multiple bits into the Interrupt register, but a new interrupt is flagged between the time software read the Interrupt register and the time it wrote the TPM_EOI.

Application Note: *Many commands respond immediately so during normal operation the driver, after sending a command, should poll the TPM for a response keeping the interrupts masked. If the driver determines that the TPM will not be able to respond immediately, it will stop polling the TPM and unmask the appropriate set of interrupts. If the driver does this, there is a possibility of a race condition between the time the interrupt is unmasked and the state being checked becomes true. Therefore, after unmasking the interrupt(s,) the driver should poll the TPM one more time.*

End of informative comment

1. The TPM MUST support the following interrupts:
    a. TPM_INT_STATUS_x.localityChangeIntOccured
    b. TPM_INT_STATUS_x.dataAvailIntOccured
2. The TPM MAY support the following interrupts:
    a. TPM_INT_STATUS_x.stsValidIntOccurred
    b. TPM_INT_STATUS_x.commandReadyIntOccured
3. The TPM MUST support asserting any of the IRQ[0:15]. The TPM MUST NOT assert PIRQ[A:D] in the SIRQ stream.
4. The TPM MUST support low level interrupts and MAY support the other interrupts. The TPM MUST report all schemes it supports in the Interrupt Capabilities.
5. The TPM MUST set the appropriate bit indicating the cause of the interrupt in the TPM_INT_STATUS_x register.
6. Once an interrupt is asserted, the TPM MUST NOT assert another interrupt until it receives a TPM_EOI even if new events occur that should cause an interrupt.
7. The software MUST select one of the schemes supported by the TPM. If the TPM supports only one scheme, bits 3 and 4 MAY be read-only and return the value of the implemented scheme.

8. The TPM_EOI to the TPM is writing a "1" to the bit in the TPM_INT_STATUS_x register that corresponds to the type of interrupt just handled. The software MAY write multiple bits if it has handled multiple interrupt causes at one time.
9. If the interrupt handler detects multiple bits set in TPM_INT_STATUS_x, it MAY handle all the causes and clear multiple status bits.
10. The TPM MUST maintain interrupts as inactive during any change to the TPM_INT_ENABLE_x globalIntEnable and while TPM_INT_ENABLE_x globalIntEnable is 0.
11. If the driver is polling and then decides to unmask interrupts, it MUST poll the TPM one more time after unmasking the interrupts.
12. The TPM has only one interrupt assigned to it. Therefore, while the interrupt ports are replicated across all locality addresses, the interrupt settings for one locality applies to all localities.
13. While polling, the driver SHOULD mask the interrupts.

12.1 Interrupt Enable

Table 20: Interrupt Enable

Abbreviation:				TPM_INT_ENABLE_x
General Description:				Enables specific interrupts and has the global enable. The TPM MUST implement this register.
Bit Descriptions:				
31	Read/ Write	globalIntEnable	Default:0	1 = Interrupts controlled by individual bits 0= All interrupts disabled. Cleared to "0" on reset.
30:8		Reserved	Reads always return 0	
7	Read/ Write	commandReadyEnable	Default: 0	1 = Enabled 0 = Disabled
6:5		reserved	Reads always return 0	Note to readers and future editors: This displacement of the enable bits (i.e. TPM_INT_ENABLE_x.commandReadyEnable not being adjacent to the other enable bits) here and in the tables below was done because the TPM_INT_ENABLE_x.commandReadyEnable bit was added late in the draft cycle of this release (1.2). Some TPM manufacturers could not change their implementation in a timely and costly manner if we moved the TPM_INT_ENABLE_x.typePolarity bits.
4:3	Read/ Write	typePolarity	Default: 01	00 = High level 01 = Low level 10 = Rising edge 11 = Falling edge
2	Read/ Write	localityChangeIntEnable	Default: 0	1 = Enabled 0 = Disabled
1	Read/ Write	stsValidIntEnable	Default: 0	1 = Enabled 0 = Disabled
0	Read/ Write	dataAvailIntEnable	Default: 0	1 = Enabled 0 = Disabled

12.2 Interrupt Status

Table 21: Interrupt Status

Abbreviation:				TPM_INT_STATUS_x
General Description:				Shows which interrupt has occurred. The TPM MUST implement this register.
Bit Descriptions:				
31:8		Reserved	Reads always return 0	
7	Read / Write 1	commandReadyIntOccured	Default: 0	When "1", indicates the TPM_STS_x.commandReady bit transitioned from 0 to 1. Writing a "1" to this bit clears the interrupt. Writing a "0" to this bit has no effect.
6:3		reserved	Reads always return 0	
2	Read / Write 1	localityChangeIntOccured	Default: 0	When "1", indicates the locality change interrupt occurred. This interrupt is caused whenever any locality moves from TPM_ACCESS_x.requestUse to TPM_ACCESS_x.activeLocality whenever this transition had been delayed due to another locality having TPM_ACCESS_x.activeLocality set. Note that if the TPM has no TPM_ACCESS_x.activeLocality set when TPM_ACCESS_x.requestUse is written, the TPM MUST move directly from TPM_ACCESS_x.requestUse to TPM_ACCESS_x.activeLocality without causing the interrupt. Writing a "1" to this bit clears the interrupt (i.e., a TPM_EOI). Writing a "0" to this bit has no effect.
1	Read / Write 1	stsValidIntOccurred	Default: 0	This interrupt indicates a "0" to "1" transition on TPM_STS_x.stsValid. Writing a "1" to this bit clears the interrupt. Writing a "0" to this bit has no effect.
0	Read / Write 1	dataAvailIntOccured	Default: 0	This interrupt indicates that TPM_STS_x.dataAvail transitioned from a "0" to a "1". This "0" to "1" transition occurs when the command has been completed and there is a Response to be read. This transition MUST only occur when both TPM_STS_x.dataAvail and TPM_STS_x.stsValid bits are "1". Writing a "1" to this bit clears the interrupt. Writing a "0" to this bit has no effect.

12.3 Interrupt Vector

1175 **Table 22: Interrupt Vector**

Abbreviation:			TPM_INT_VECTOR_x
General Description:			Contains the SIRQ value. The TPM MUST implement this register.
Bit Descriptions:			
7:4	Reserved (0)	Reads always return 0	Must return "0" on read; writes have no meaning.
3:0	sirqVec	Default: 0	A value of "0" means SIRQ is disabled and SIRQ is tristated. The SIRQ channel used by TPM can be from 1 to 15.

13. TPM Interface: LPC

Start of informative comment

The LPC bus is the assumed connection of the TPM to the chipset. This section covers the implementation issues for that connection.

End of informative comment

13.1 Existing TPM Cycles

Start of informative comment

The existing implementations of the TPM use LPC I/O cycles. I/O cycles are only defined to support a 16-bit address space. The TPM may continue to use its legacy port.

The existing port has a Locality of 0 as regards its capabilities with respect to the PCR registers but has a Locality of –1 with respect to the way the TPM_ACCESS_x register works.

End of informative comment

The TPM MAY continue to use its legacy port.

13.1.1 Extension to Existing TPM Cycles

Start of informative comment

The TPM 1.1 has ports in I/O space for legacy software to use. That legacy software is unaware of the new REQUEST and SEIZE functions. To support the legacy software, if the TPM supports legacy ports, the TPM must accept cycles on the I/O ports any time that no TPM_ACCESS_x.activeLocality bit is set. The TPM must not accept cycles on the I/O ports if any locality is set.

The use of legacy ports is specified in Section 9.4.

End of informative comment

13.2 New LPC Locality Cycles for TPM Interface

Start of informative comment

Two new LPC special cycles, TPM-Write and TPM-Read, are added for communication between the chipset and the TPM. On the LPC bus, with the exception of the START field, these cycles are identical to I/O cycles. These new cycles are an additional flag to the TPM (beyond addressing) that the cycles are intended for the TPM as the new locality commands.

End of informative comment

See Section 9.1 for rules and restrictions on using the standard vs. Locality LPC cycles.

13.2.1 TPM-Write LPC Locality Special Cycle

Start of informative comment

Table 23 shows the new TPM-Write special cycle format. It is similar to the existing LPC I/O write.

If the CPU attempts to write more than 1 byte at a time to the TPM, the chipset must break this up into multiple cycles of 1 byte each to consecutive addresses.

End of informative comment

Table 23: New LPC Special Cycle TPM-Write for Accessing the TPM

Field	Value for Bits [3:0]	Description
START	0101	Previously this was a reserved value. It is now allocated for TPM-Write and TPM-Read.
CYCTYPE + DIR	0010	Same as used for standard LPC I/O Write
ADDR	See Description	Four nibbles. Same as the standard LPC I/O Write.
DATA-Low	DIGEST low nibble	
DATA-High	DIGEST high nibble	
TAR		Standard LPC TAR
SYNC		Standard SYNC field for an I/O Write
TAR		Standard LPC TAR

13.2.2 TPM-Read LPC Locality Special Cycle

Start of informative comment

Table 24 shows the new TPM-Read special cycle format. It is similar to the existing LPC I/O read.

If the CPU attempts to read more than 1 byte at a time to the TPM, the chipset must break this up into a series of 1-byte reads to consecutive addresses.

End of informative comment

Table 24: New LPC Special Cycle TPM-Read for Accessing the TPM

Field	Value for Bits [3:0]	Description
START	0101	Previously this was a reserved value. It is now allocated for TPM-Write and TPM-Read.
CYCTYPE + DIR	0000	Same as used for standard LPC I/O Read
ADDR	See Description	Same as for TPM-Write
TAR		Standard LPC TAR
SYNC	Standard	Standard SYNC field for an I/O Read
DATA-Low	DIGEST low nibble	
DATA-High	DIGEST high nibble	
TAR		Standard LPC TAR

13.3 TPM Byte Ordering

Start of informative comment

The TCG specification has multi-byte fields, defined as big-endian. To meet the big-endian requirement for multi-byte fields, the TPM must first receive the MSB (most significant byte) on the LPC bus. For instance, the TCG_PROTOCOL_ID is 0x0005 for PID_OWNER. On the LPC bus, this must be sent to the TPM with 0x00 as the first byte and 0x05 as the second byte. However, the bridge between the CPU and the TPM is little-endian based and, therefore, a CPU write of 0x0005 would send the 0x05 to the TPM first, then the 0x00 which is not the correct sequence for the TCG_PROTOCOL_ID for PID_OWNER.

Software is required to handle the big-endian TPM fields vs. the little-endian nature of the bridge and LPC bus. Software may do Double Word (DW) (4 bytes) or Word (2 bytes) or Byte accesses to the TPM. Standard PC platforms will have bridges that translate these 4-byte or 2-byte accesses into single-byte accesses on LPC. PC platforms will always break up the request so that they send the least significant address of that request first on the LPC bus. In the example above, software could do two 1-byte accesses to the TPM, the first write would be with data 0x00, and the second write would be with data 0x05. Or software could do a Word access and send 0x0500, which would cause the bridge to issue the 0x00 cycle first on the LPC bus followed by an LPC cycle of 0x05.

End of informative comment

The TCG specification has multi-byte fields, defined as big-endian; to meet the big-endian requirement for multi-byte fields, the TPM MUST first receive the MSB (most –significant byte) on the LPC bus.

Software MUST handle the big-endian TPM fields vs. the little-endian nature of the bridge and the LPC bus. Software MAY do Double Word (DW) (4 byte) or Word (2 byte) or Byte accesses to the TPM.

13.4 Reset Timing

Start of informative comment

The operation of the platform's CRTM likely occurs during a very time-sensitive period. Because of this, strict requirements are necessary for the TPM's reset timing. During this time platform's CRTM may need to make decisions based on the presences or absences of the TPM's response that affect the rest of the platform's boot cycle. This requires the any return from TPM_ACCESS_x register be valid regardless of the timing – the TPM must not be allowed to return anything but a valid response from this register.

While the TPM_ACCESS_x register is the most critical, the availability of the other registers is important for performance reasons.

End of informative comment

1. The TPM MUST return a valid response to a read of any TPM_ACCESS_x register within 100 μs after completion of TPM_Init.
2. The TPM MUST NOT return a response to any read of any of the TPM_ACCESS_x registers until the register is valid.

3. All other registers MUST return a valid response to reads within 30 ms after completion of TPM_Init.
4. The TPM MUST accept commands and MUST have set the TPM_ACCESS_x.tpmRegValidSts bit within 30 ms after completion of TPM_Init.

13.5 Other LPC Considerations

Issue	**Description**
Mobile	The TPM is permitted to use the CLKRUN# protocol for mobile platforms.

14. TPM Packaging

Start of informative comment

A standard package allows TPM and motherboard manufacturers the convenience and cost savings of not having to define from scratch the packaging and pinout for a TPM. This packing and pinout standard is provided as a convenience for either an end product or as a basis for extension or modification. It is recognized that individual environments may dictate other schemes; therefore, implementation of this section is optional and any deviance will not detract from a platform's claim to adherence to this specification.

End of informative comment

The TPM MAY use the packaging and pinout definitions as defined in this section and per Figure 2.

The TPM MAY be designed as a 28-pin TSSOP using 9.6 mm (0.65 mm lead pitch) 6.1 mm plastic width.

If a TPM is designed to use the packaging and pinout outs specified in this section, it SHALL be said to use the "Packaging as specified in the TCG PC Client Specification 1.2". If it does not use this pinout and packaging, it MUST not claim this packaging.

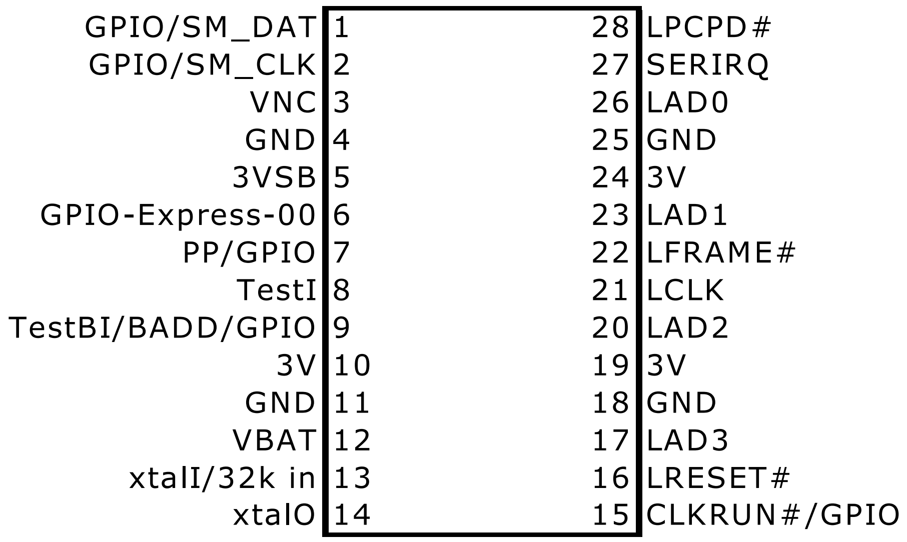

Figure 2: TPM Pinout

Using Figure 2, the pins MAY be defined as in Table 25.

Table 25: Pin Assignments

Signal	Pin(s)	Type	Description	Required
LAD[3:0]	26, 23, 20, 17	BI	As defined in the LPC Interface Specification	Y
LPCPD#	28	I	As defined in the LPC Interface Specification	Y
LCLK	21	I	As defined in the LPC Interface Specification.	Y
LFRAME#	22	I	As defined in the LPC Interface Specification	Y
LRESET#	16	I	As defined in the LPC Interface Specification	Y
SERIRQ	27	BI	As defined in the LPC Interface Specification	Y
CLKRUN#/GPIO	15	BI	Same as PCI CLKRUN#. Active Low. Only needed by peripherals that need DMA or bus mastering in a system that can stop the PCI bus. (Generally Mobile) GPIO will default to low.	Y
PP/GPIO	7	I,BI	Physical Presence, active high, internal pull-down. Used to indicate Physical Presence to the TPM. GPIO will default to low	Y
XTALI/32k in	13	I	32 kHz crystal input or 32 kHz clock input	
XTALO	14	O	32 kHz crystal output	
GPIO/SM_CLK	2	BI	Defaults as a GPIO. GPIO will default high. Also used as System Management Bus Clock signal	
GPIO/SM_DAT	1	BI	Defaults as a GPIO. GPIO will default high Also used as System Management Bus Data signal.	
GPIO-Express-00	6	BI	GPIO assigned to TPM_NV_INDEX_GPIO_00	
VNC	3		Vendor Controlled No Connect. This pin will be defined by the TPM vendor or can be a GPIO. There is no defined default state for this signal.	
TESTBI	8	I	This pin will be pulled low on the motherboard. Assuming: Pull high to enable Test mode. Pull low to disable Test mode and enable GPIO on pin 8(TESTBI).	
TESTBI/ BADD/GPIO	9	8	TESTBI: Test port. If TESTBI is pulled low acts as a GPIO. BADD(Optional): Used to switch between addresses. Internal pull-up. Status read at startup. High = 7E/7F (default) Low(external pull-down) = EE/EF GPIO will default high	
Power				
3V	10, 19 24	I	This is a 3.3 volt DC power rail supplied by the motherboard to the module. The maximum power for this interface is 250 mA. Available from S0-S2.	Y
GND	4, 11, 18, 25	I	Zero volts. Expected to be connected to main motherboard ground.	Y
VBAT	12	I	3.3 V battery input. Available from S0-S5 and in G3 state.	
3VSB	5	I	3.3 V standby DC power rail. Available from S0-S5.	

15. References

1. The Trusted Computing Group: http://www.trustedcomputinggroup.org
2. Low Pin Count (LPC) Interface Specification: http://www.intel.com/design/chipsets/industry/lpc.htm
3. System Management Bus (SMBus) Specification Version 2.0: http://www.smbus.org
4. PCI Express Electromechanical Specifications http://www.pcisig.com
5. The Serial IRQ (SERIRQ) protocol definition is at http://www.smsc.com/ftpdocs/papers.html