TCG PC Client Specific Implementation Specification For Conventional BIOS

Version 1.20 FINAL
Revision 1.00
July 13, 2005
For TPM Family 1.2; Level 2

Change History

Revision	Date	Description
1.00	July 13, 2005	• Initial release of Version 1.20.

Contents

Figures

Tables

Corrections and Comments

TCG members may send comments to:

techquestions@trustedcomputinggroup.org

TPM Dependency and Requirements

1. 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.
2. The TPM used for Host Platforms claiming adherence to this specification MUST be compliant with the *TCG PC Client Specific TPM Interface Specification; Version 1.2; Revision RC29*.
3. The platform Class for platforms adhering to this specification SHALL be registered with the TCG Administrator.[1]

[1] This value is neither returned, nor used, by any entity defined in this specification. This value is normative to software, policy engines, etc., that need to define how to interpret information from platforms adhering to this specification.

1. Introduction and Concepts

Start of informative comment

The Trusted Computing Group's architecture is a platform independent architecture to enhance trust on computing platforms. As such, the TCG Main Specification is general in specifying both hardware and software requirements. The goal of the TCG member companies is to ensure compatibility among implementations within each computing architecture. It is expected that companion implementation documents will be created for each architecture.

This document serves as implementation reference document for the 32-bit PC architecture. Specifically, this document defines:

1. Usage of PCR registers in the Pre-Operating System State through the transition to Operating System Present State.
2. How the BIOS, or a component thereof, functions as the Core Root of Trust for Measurement (CRTM).
3. Programmatic interfaces to the BIOS as it performs the functions of the TCG Subsystem (TSS and access to the TPM).
4. Behavior entering, during, and exiting power and initialization states.
5. Guidelines for Option ROMs.

This specification is based on the *TCG Main Specification Version 1.2*. The reader is expected to have an understanding of the concepts, defined functionality, and terms expressed in that document. This specification will attempt to minimize the duplication of information from that document; therefore, concepts and terms defined in the TCG Main Specification will not be defined in this document. If there is a conflict in interpretation between this and the TCG Main Specification, the concept or functional description as defined in the TCG Main Specification will take precedence.

This specification also references other specifications as listed in Section 1.6. The reader is expected to be familiar with the concepts and terminology contained in each specification where relevant.

It is important to understand that there are two uses for measurements: Attestation and Sealed Storage.

1. Attestation is used to provide information about the platform’s state to a challenger. However, PCR contents are difficult to interpret; therefore, Attestation is typically more useful when the PCR contents are accompanied by a measurement log. While not trusted on their own, the measurement log contains a richer set of information than does the PCR contents. The PCR contents are used to provide the validation of the measurement log.
2. Sealed Storage uses only the PCR contents to determine its action. Sealed Storage operations make no use of the measurement log and are simpler but provide only a success or fail for the validation of the platform’s configuration.

If the only use of PCRs were Attestation, there would be little reason to have more that just a few PCRs because they are only used for the validation of the measurement log. Challengers could validate the log, then parse through the measure log for the information they need. Sealed Storage is best done when the types of measured objects are grouped with objects of similar impact to the validation of the platform's trust grouped into the same PCR. This logic was chosen when arriving at the PCR mapping in this specification.

End of informative comment

1.1 PC Client Specific Architecture

Start of informative comment

The concepts and descriptions of the PC architecture are presented in both Figure 1 and the subsequent descriptions. While Figure 1 infers physical connections, the connections and associations between the components are logical.

End of informative comment

Figure 1 depicts the general architectural components of the PC Client as used in this specification. The components and their relationships within the diagram are meant to provide a reference for discussion and are not meant to require or imply any particular design or implementation beyond what is stated in the normative text in this specification.

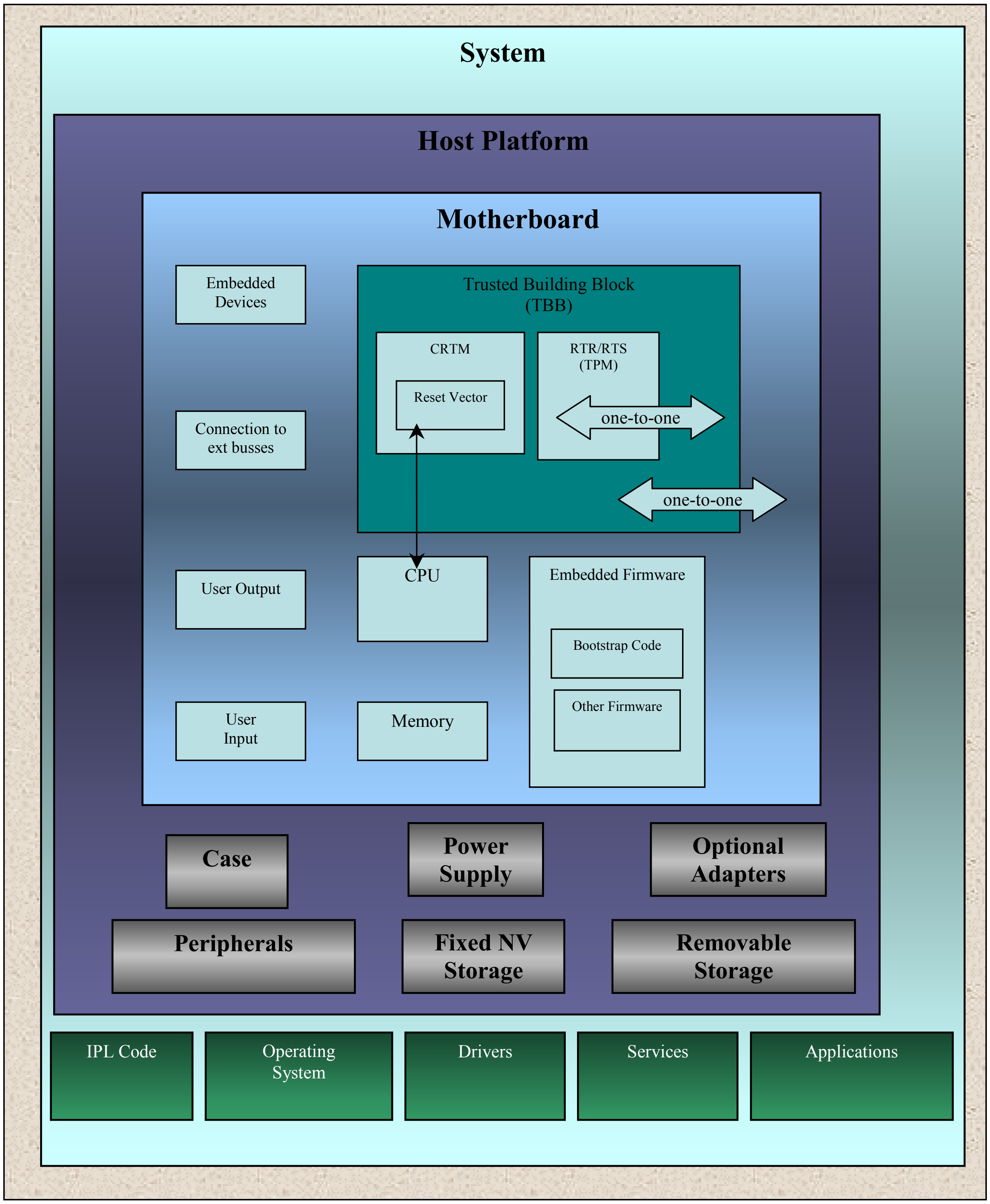


Figure 1: PC Client Architectural Components

1.2 PC Client Concepts

1.2.1 Host Platform TPM

The term TPM within this specification SHALL refer to the TPM attached to the Host Platform for the purpose of providing protected capabilities for the Host Platform as defined by the TPM Specification identified in the TPM Dependencies and Requirements section above.

1.2.2 Immutable

In this specification, immutable means that in order to maintain trust in the Host Platform, the replacement or modification of code or data MUST be performed by a Host Platform manufacturer-approved agent and method. This allows a manufacturer to establish an upgrade method for the portion of the Host Platform which is the CRTM with consideration of the security properties of the Platform's Protection Profile.

1.2.3 Trusted Building Block (TBB)

The Trusted Building Block (TBB) is the combination of the CRTM, TPM, connection of the CRTM to the motherboard, and the connection of the TPM to the motherboard. The connection of the CRTM to the TPM is done through transitive trust of the CRTM connection and the TPM connection.

Since the CRTM and the TPM are the only trusted components of the motherboard and since indication of physical presence requires a trusted mechanism to be activated by the Host Platform Owner, the indication of physical presence MUST be contained within the TBB.

1.2.4 Host Platform

The Host Platform is the entity that executes the Host operating system, which executes, presents output from, and receives input for the Host Applications for the users (remote or local). The Host Platform includes the motherboard, the Host Platform's CPU, the Host RTM, the Host TPM, and all the Host Peripherals that are attached[2] to the motherboard. The Host TPM's Endorsement Key represents the identity of the Host Platform.

1.2.5 Non-host Platforms

A Non-host Platform is a self-contained execution environment within the system. These platforms execute in an environment separate from the Host Platform components. This is not to be confused with a peripheral, which while potentially containing a powerful engine, only services and responds to requests from the Host Platform. For example, an IPMI compliant management controller in servers would be considered a Non-host Platform. The Non-host Platform MUST NOT prevent the measurement, recording, and reporting of the

[2] Primary peripheral device refers to devices which directly attach to and directly interact with the CPU. Examples are PCI cards, LPC components, USB Host controller and root hub, attached serial and parallel ports, etc. Examples of devices not included in this class are USB and IEEE 1394 devices.

true state of the platform. If a Platform Credential or Security Target is produced for this platform, it SHOULD provide an indication that a Non-host Platform exists.

1.2.6 System

The System includes the platform and all the Post-boot components that comprise the entire entity that performs actions for, or acts on behalf of, the user. It is the entity that is the union of the Host Platform and the Non-host Platforms. The Host Platform and the Non-host Platforms may affect and influence each other.

1.2.7 Host Platform Reset

Start of informative comment

This is an event that causes execution of the Host Platform's CPU to end its current instruction sequence and begin at a predetermined location, without the ability to return directly to a previous state. A Host Platform Reset causes all Host components to behave in their default, power-on state.

This can be caused by several events, including those in the following non-exclusive list: Initial Power-On; activation of a hardware reset line (i.e., PCI_Reset) including activation of the TPM_Init signal, initiated by the operating system to begin a new boot session, and initiated by the CPU during certain unrecoverable fault conditions. The Host Platform is to have a consistent behavior, from a trust perspective, regardless of the cause of the reset.

This section references only resets that apply to the Host Platform. Resets for Non-host Platforms and Systems are outside the scope of this specification.

Host Platform Reset only deals with establishing trust in the CRTM, not with other Host Platform components. Since all Host Platform components that are part of the transitive trust chain are measured, the action taken, or lack thereof, by these components to a Host Platform Reset has no impact on the validity of the transitive trust chain. There may be an impact on the verifier's trust in the system but that is outside the scope of this specification.

Of primary concern for establishing the transitive trust chain is that the reset of the Host Platform's CPU which causes execution to begin within the S-CRTM is "effectively simultaneous" with the Host TPM's reset.

End of informative comment

1.2.7.1 Types

A **Cold Boot Host Platform Reset** occurs when transitioning the Host Platform from a full Power-Off state in which no operating system-specific state or status is preserved on the Host Platform except for that which is contained on any operating system load device to a Power-On state. This excludes returning from various power or suspend states which can occur after the Cold Boot Reset from an operating system present state.

A **Hardware Host Platform Reset** occurs when a signal activates the reset signal of all Host Platform components. This may be a user-initiated event or a software-initiated event triggered by a command to a hardware component that asserts the reset line.

A **Warm Boot Host Platform Reset** occurs when software (often caused by a user keyboard input but may be software induced) causes a Host Platform Reset.

1.2.7.2 Host Platform CPU(s) and Components

1. Upon a Host Platform Reset, the Boot Strap Host Platform CPU MUST be reset and begin execution within the S-CRTM.
2. All remaining Host Platform CPU(s) MUST be reset.

1.2.7.3 TPM Reset

1. The TPM MUST NOT be reset without a Host Platform Reset.
2. The TPM MUST be reset (i.e., execution of TPM_Init) when the Host Platform is reset.

1.2.8 PCI Option ROM Request for Reset

Upon return from a PCI Option ROM, the BIOS MUST check the return status from the Option ROM and if requested, the BIOS MUST perform a Host Platform Reset per Section 5.2.1.24.1 of the *PCI Firmware Specification, Revision 3.0*.

1.2.9 Trusted Process

This is either a hardware-based or software-based process with the Host Platform that is trusted without the need for further inspection to perform as expressed by the Host Platform Certificate. The trusted process is the RTM for that trust domain (e.g., Static RTM or Dynamic RTM).

1.2.10 Roots of Trust (RTM)

The terms RTM and CRTM within this specification SHALL refer to those entities as they are associated with the Host Platform.

1.2.10.1 Root of Trust for Measurement (RTM)

The RTM is axiomatically trusted. Trust in this component is expressed in the Host Platform Certificate. This is the point from which all trust in the measurement process is predicated. The RTM includes a core component (the CRTM), the computing engine to run the core component, and the physical connections of the core and the computing engine.

1.2.10.2 Core Root of Trust for Measurement (CRTM)

The component of the RTM from which the platform begins execution of one of its trusted states. Each transitive trust chain is rooted at this point.

1.2.10.3 Privacy Setting and the Scope of the RTM

Start of informative comment

If the Host Platform implements privacy settings using the command method for the indication of physical presence, those settings must be under control of a process within one of the chains of trust. This is to allow verifiers (including the user or operator) a method to validate that their privacy settings are respected and enforced.

End of informative comment

1. Any privacy setting that uses the command method for an indication of physical presences MUST be measured as part of that domain's RTM.

1.2.11 Boot State Transition

The transition between Pre-Operating System State and Operating System Present State is the first invocation of INT 19h or equivalent.

1.2.12 Establishing the Chain of Trust

1.2.12.1 Bindings

1.2.12.1.1 Bindings Between an Endorsement Key, a TPM, and a Host Platform

The relationship between the Endorsement Key, a TPM, and a Host Platform is described in the *TPM Main Specification, Part 1, Design Principles*, Section 11.2.

1.2.12.1.2 Binding Methods

Start of informative comment

The method of binding the TPM to the motherboard is an architectural and design decision made by the respective manufacturer and is not specified here. There are two types of binding: physical and logical. Physical binding relies on hardware techniques while logical binding relies on cryptographic techniques. The nature and strength of each method is defined by the TPM's or the Platform's Protection Profile.

Example:

The TPM is a physical chip soldered to the Host Platform. Here the Endorsement Key is physically bound to the TPM (it's inside it) and the TPM is physically bound to the Host Platform by the solder. The required strength of each binding is determined by the Protection Profile.

End of informative comment

1.2.13 Locality States

Start of informative comment

A general description of locality can be found in the *TPM Main Specification, Part 1, Design Principles*, Section 14.

In TCG-enabled Version 1.1 platforms, there was only one CRTM. It started at Host Platform Reset. This architecture causes some large constraints within some operating environments because all components of the Host Platform must participate in the chain of trust. Locality provides an expression of a CRTM that is not dependent on the Host Platform Reset called the Dynamic CRTM (D-CRTM). As described in Figure 2, these two RTMs and their respective chains of trust have no relationship to each other except that they are both rooted at the same RTS/RTR (i.e., the TPM). This is an important consideration because the trust of each is dependent only on the trust in the common RTS/RTR and their own RTM.

End of informative comment

1.2.13.1 Locality State Definition

1. Locality is an expression of the execution of an RTM.
2. Locality must be asserted by a trusted process: either hardware or software.
3. Locality is asserted to the TPM by the Host Platform's trusted processes by the LOCAL_MOD = TRUE signal.
4. Trust in the trusted process that asserts locality is expressed in the Host Platform Certificate.

1.2.13.2 Locality State Relationship

Start of informative comment

The expression of trust for each RTM is independent of the other. That is to say, each trust statement is verifiable within its own domain rooted at its own respective RTM. However, each Chain of Trust exists within an IT environment that is likely dependent on other components. For example, the trust in a Host Platform, which is verifiable within its own domain is dependent on assumptions about its environment such as the routers are valid, the physical protections are in place, etc. It is possible and even conceivable that trust in the Host Platform as an entity will rely on the trust in the D-RTM, C-RTM, or some subset of both.

For example, the Host Platform resides within a political region that requires certain privacy controls be respected and enforced prior to allowing the Host Platform to participate in using IT resources. The CRTM may not be able to verify and, therefore, assert that the user had proper use and control of the Host Platform's privacy settings. In this case, the IT infrastructure may require that the privacy setting be controlled and asserted by the processes within the S-RTM's chain of trust. Therefore, when the Host Platform boots and attempts to join the IT environment, the verifier will first verify the chain of trust associated with the privacy setting (i.e., the S-RTM’s chain of trust) before proceeding to verify the security assertions of the D-RTM.

End of informative comment

1. The only commonality between the D-RTM and the S-RTM is the RTR/RTS (TPM).
2. There is no direct relationship between, or dependence on, the trust in the chain of trust established by the D-CRTM and the chain of trust established by the S-CRTM. From a trust perspective, these are architecturally distinct entities. Specific designs MAY create a dependency between them however. This dependency, if any, SHOULD be represented in the Host Platform Certificate.
3. However, external entities and verifiers MAY associate the two chains of trust as being part of the same Host Platform where the two chains coexist within the Host Platform and, therefore, are treated at least in part as a union of their trust statements within a larger environment.

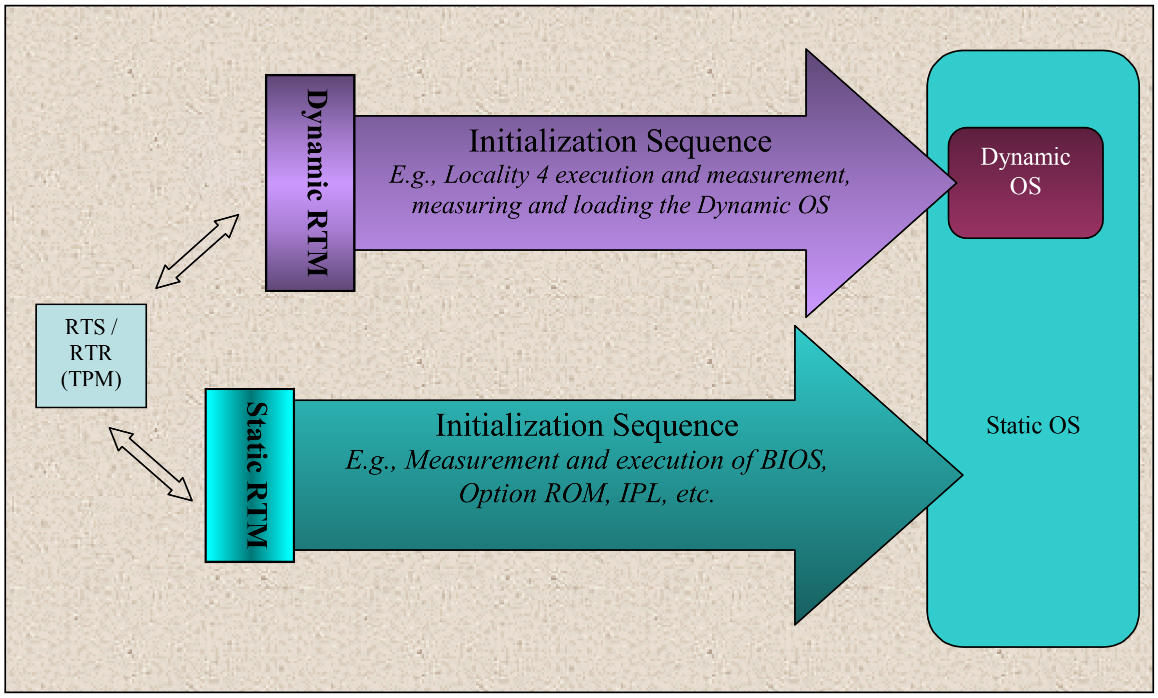

Dynamic RTM
Initialization Sequence
E.g., Locality 4 execution and measurement, measuring and loading the Dynamic OS
Dynamic OS
RTS / RTR (TPM)
Static RTM
Initialization Sequence
E.g., Measurement and execution of BIOS, Option ROM, IPL, etc.
Static OS

Figure 2: Relationship Between Static and Dynamic RTMs

1.3 Overview of the Measurement Process

Start of informative comment

This section uses, but does not define, concepts that are explained later in this specification. Therefore, to understand this process and sequence, the reader should be familiar with the remainder of this specification.

The general sequence of operations is diagramed in Figure 3. While the sequencing in this diagram is described in the normative within this specification, the diagram is informative only. No attempt is made in this general description to itemize the optional and mandatory components or sequences.

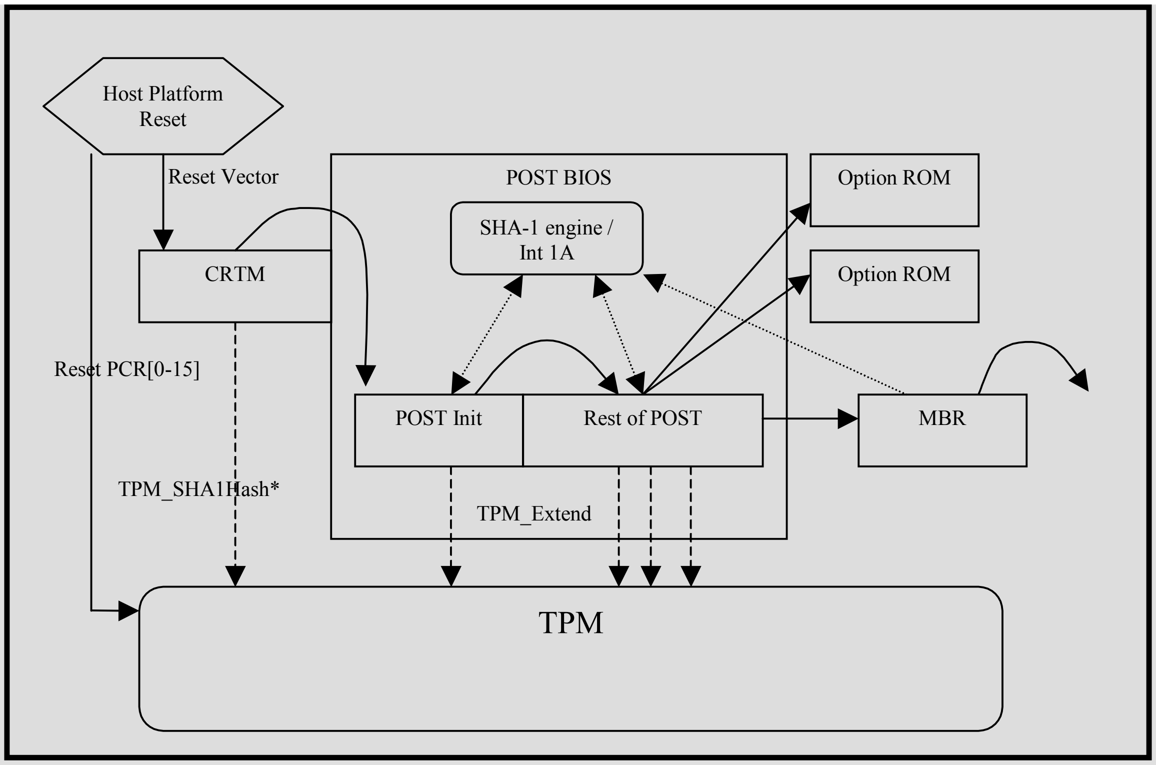


Figure 3: Example Boot Flow

End of informative comment

1.3.1 Usage and Optimization of Hash Functions

Start of informative comment

Among the set of functions provided by the TPM are a set of hash functions: TPM_SHA1Complete, TPM_SHA1CompleteExtend, TPM_SHA1Start,; and TPM_SHA1Update.

These functions are provided for environments were implementing a hash function would be either impossible or very slow. One example is within an implementation of the S-CRTM where memory is not yet enabled. In this implementation of this environment, it would be faster to send the data to be hashed to the TPM rather than have even a fast Host Platform CPU perform this function using only registers.

However, it must be kept in mind that the LPC bus and the TPM are comparatively slow devices and judicious use of these TPM-based hashing functions is prudent. It is best to utilize these for as few measurements as possible and only until memory is enabled then the BIOS should switch over to using its own hashing function as quickly as possible, thereby improving boot time performance.

The goal of utilizing the TPM's hashing functions as little as possible is demonstrated in this example and in the diagram above. The Host platform reset causes the set of static PCRs (i.e., PCR[0-15]) to be reset to their default values (i.e., 0). It also causes the Host Platform CPU to begin executing at the reset vector, which is within the S-CRTM. The S-CRTM's operational environment typically has functionality and is unlikely to have memory available. Thus, it must use the Host Platform CPU's internal registers only. Performing a hash operation within such a limited environment would be very time consuming even for the very fastest CPUs. For this reason, it is recommended that S-CRTM utilize the TPM_SHA1* commands. While the use of the TPM's hashing capabilities improves performance during the non-memory environment, the Host Platform CPU is considerably faster than the TPM once memory is available. It is, therefore, recommended to measure and enable a Host Platform-based SHA-1 hashing engine as quickly as possible.

In the example provided in Figure 3, the S-CRTM uses the TPM's set of TPM_SHA1* functions to measure the smallest component of the POST code. If architected in such a way to allow partitioning of the POST's security components, the S-CRTM would measure only the portion of the POST that is called Post Init in the diagram. The only requirement of the POST Init module is that it measures any component prior to jumping to it. In this architecture, the POST Init module could perform the minimum functions necessary to enable memory, initialize a memory present SHA-1 engine, and then use that SHA-1 engine to perform the subsequent measurements. This satisfies the "Chain of Trust" rule because the POST Init module was measured prior to enabling of memory and the SHA-1 engine.

End of informative comment

1. The BIOS MAY use the TPM's hashing commands (i.e., TPM_SHA1*) but SHOULD do so for as little data and as few times as possible.
2. The BIOS SHOULD use its own Host Platform CPU-based hashing function as early as possible and SHOULD continue to use it.

1.4 PC Client-specific Definitions

Start of informative comment

These definitions are in alphabetical order because of some cyclic definition dependencies.

End of informative comment

1.4.1 BIOS Recovery Mode

Start of informative comment

This is a failure-recovery mode of the BIOS that is invoked by the BIOS Boot Block typically when the main BIOS is corrupt. See Section 6.1.

End of informative comment

1.4.2 Host Platform CPU

Start of informative comment

The computing engine(s) of the Host Platform. Multiple CPUs may be contained on a single Host Platform but, for the purposes of this specification, are treated as a single unit. In a multiple CPU Host Platform, each CPU must perform reset and initialization as defined in Host Platform Reset. Host Platforms containing multiple CPUs are assumed to load and execute the same Operating System. For the purpose of the remainder of this specification, the term CPU will refer to all CPUs on the Host Platform.

End of informative comment

1.4.3 Core RTM (CRTM)

Start of informative comment

The executable component of the RTM that gains control of the Host Platform upon a Host Platform Reset. See a more detailed description of this in Section 3.2.1.2.

End of informative comment

1.4.4 Dynamic Operating System

Start of informative comment

The operating system that is dynamically loaded sometime after and usually at the initiation of the Static Operating System. There may be more than one Dynamic Operating System per Host Platform, but only one can be loaded at a time. The Dynamic Operating System can be unloaded keeping the Static Operating System resident and operational.

End of informative comment

1.4.5 Initial Program Loader Code (IPL Code)

Start of informative comment

This area of the IPL Image contains only the code that executes during the Post-boot state. The purpose of this code is to load the Post-boot environment.

End of informative comment

1.4.6 Initial Program Loader Image (IPL Data)

Start of informative comment

This area of the IPL Image contains only data. For example, this area contains the MBR's partition table.

End of informative comment

1.4.7 Initial Program Loader Image (IPL Image)

Start of informative comment

This area contains the IPL Code and any data. An example of an IPL Image is the first section of an hard disk's MBR. This area contains both the executable IPL Code and the partition table.

End of informative comment

1.4.8 Measurement and Measure

Start of informative comment

These terms mean to perform hash, log, and extend to the appropriate PCR(s).

End of informative comment

1.4.9 Manufacturer

Start of informative comment

The entity that makes and attests to the validity of a component. In this specification, unless otherwise noted, this refers to the maker of the TBB. Unless otherwise stated, this refers to the manufacturer of the motherboard.

End of informative comment

1.4.10 Motherboard

Start of informative comment

An entity that is supplied by the manufacturer which is comprised of the TBB and other components physically or logically attached and supplied by the manufacturer.

End of informative comment

1.4.11 Operating System Present State

Start of informative comment

The state of the system after the invocation of the first INT 19h or its equivalent. This may include operating system, PARTIES, diagnostics, etc. Contrast to *Pre-Operating System State*.

End of informative comment

1.4.12 Pre-Operating System State

Start of informative comment

The state of the system prior to the invocation of the INT 19h or its equivalent. Contrast to *Operating System Present State*.

End of informative comment

1.4.13 Static Operating System

Start of informative comment

The operating system that is loaded during the initial boot sequence of the platform from its platform reset. Typically, when the Static Operating System is unloaded, the platform performs a platform reset.

End of informative comment

1.4.14 Trusted Building Block (TBB)

Start of informative comment

The combination of the CRTM, TPM, connection of the CRTM to the motherboard, and the connection of the TPM to the motherboard. See Section 1.2.3.

End of informative comment

1.5 TCG Specification Dependency and Naming

The following TCG Specifications are referenced in this specification.

1.5.1 "This" Specification

References to "this specification", unless contextually referencing a different antecedent, refers to the informative and normative comments contained in this document. That is, *TCG PC Client Specific Implementation Specification for Conventional BIOS; For TCG, Version 1.2* as released."

This specification defines only functional aspects of the concepts and implementing a TPM, RTM, and other support features for a PC Client Platform. The definition of the security mechanisms and the strength of those mechanisms are intentionally outside the scope of this specification.

1.5.2 TPM Interface Specification (TIS)

The *TPM Interface Specification* (TIS) is considered the "companion" specification to this one. The TIS defines the interface to the TPM and how it is connected (i.e., which local Host Platform bus). The target audience for the TIS is the TPM developers and all level of driver writers.

1.5.3 Protection Profile

As stated above, this specification defines only functional behaviors. All security requirements including the necessary strength of those security requirements are defined in a companion specification (the related TBB for Conventional BIOS Protection Profile).

1.5.4 TCG Architecture, Version 1.2

Start of informative comment

Refers to the *TCG Architecture, Version 1.2* as released by the Trusted Computing Group. For the remainder of this document this shall be referred to as the TCG Architecture unless otherwise noted.

End of informative comment

1.5.5 TCG Main Specification

Start of informative comment

Refers to the *TCG TPM Main Specification, Version 1.2* as released. There are four parts to this specification. Unless a specific part is referenced, this reference refers to all parts of the specification.

End of informative comment

1.5.6 TCG TSS Specification

Start of informative comment

Refers to the *TCG TSS Specification, Version 1.2* as released.

End of informative comment

1.6 External Specifications

This section lists references to external non-TCG specifications.

1.6.1 Plug and Play BIOS Specification

Version 1.0A.

1.6.2 Advanced Configuration and Power Interface Specification

Revision 2.0, July 27, 2000.

Referred to as “ACPI” in this specification.

1.6.3 BIOS Boot Specification

Version 1.01, January 11, 1996.

1.6.4 Boot Integrity Services Application Programming Interface

Version 1.0.

1.6.5 System Management BIOS Reference Specification

Specification number dsp0134 at: http://www.dmtf.org/standards/published_documents

Referred to as “SMBIOS” in this specification.

1.6.6 “El Torito” Bootable CD-ROM Format Specification

Version 1.0, January 25, 1995.

1.6.7 Preboot Execution Environment (PXE) Specification

Version 2.1.

1.6.8 PARTIES (Protected Area Run Time Interface Extension Services)

Working Draft, T13 D1367, Revision 3, September 30, 2000.

2. Host Platform Setup and Configuration

2.1 Pre-Operating System State ROM-based Setup

If this utility changes any component that has been measured upon completion, this setup utility MUST perform a Host Platform Reset. This includes setup utilities provided by either the motherboard-based BIOS and Option ROMs.

Entry into this state MUST be measured as event "Entering ROM-based Setup".

2.2 Post-boot ROM-based Setup

Start of informative comment

This is ROM-based setup accessed via a keyboard hot-key during the post-boot state.

End of informative comment

The setup utility MUST NOT allow changes to the components of the Host Platform configuration that are already measured unless the Post-boot environment can measure the event or the setup utility provides a mechanism to notify the Post-boot Operating System that a change occurred.

2.3 Reference Partition and Operating System-Based Utility

A reference partition is defined as a component that is loaded and executed by the INT 19h or INT 18h handler. This is treated as IPL Code. The setup utility within the reference partition MUST measure events that affect Host Platform configuration.

3. Localities

3.1 Summary

Start of informative comment

General definition of Locality:

In TPM, Version 1.1, there was only one transitive trust chain and, therefore, only one RTM, i.e., the Host Platform's BIOS. The *TPM Main Specification,* Version 1.2 allows for the definition of multiple transitive trust chains. These transitive trust chains are identified using a concept called locality.

An RTM begins in a known and trusted privileged execution state and environment. This is necessary to establish trust in an execution environment's transitive trust chain. Locality is a mechanism that associates an RTM (i.e., the root of a transitive trust chain) to a TPM.

Besides associating a locality with a specific RTM and a specific transitive trust chain, a locality is associated with a set of PCRs or other objects with the TPM (e.g., keys, NV storage, etc.). Thus, locality binds a specific execution environment to a specific set of PCRs or other TPM objects. Locality also associates a command with its execution environment and to the execution environment's privilege level.

Localities may be grouped with a set of localities associated with a single transitive trust chain. These localities may be hierarchical with some being associated with higher privileged level or execution environments than the others within the set.

The reset mechanism for an RTM must provide the equivalent to access control over the entities within the platform that have access to the PCRs (and other TPM objects) associated transitive trust chain. These associated PCRs and other TPM objects are all defined by the platform specific specification (this set of specifications).

It is important to consider that the TPM itself provides no access control between the execution environment and locality; therefore, locality's access control point is outside the TPM. The TPM, therefore, assumes that the expression of any locality is from the associated execution environment.

There is no requirement for a platform's architecture to support more than one locality. It is, however, meaningless to speak of locality on a platform that contains only one – as the one is always implied. However, on platforms that contain more than one, it is significant to indicate which locality is being used or reference. In the PC Client, there are two sets of localities: Locality 0 and Locality 1-4. There is a hierarchy within the locality set 1-4, therefore, four privilege levels; but Locality 0 has no other localities associated with it; therefore, there are is only one privileged level associated with Locality 0.

This section provides an overall view of the PCRs and their relationship to each transitive trust chain. The normative and detailed description of the various PCR attributes is defined in the *PC Client TPM Interface Specification.*

End of informative comment

Table 1 is a summary of the Locality usages.

Table 1: Locality Usage

Locality	Entity in Control	PCRs that can be Reset by the Locality	PCRs that can be Extended by this Locality
Any	Any Software	16, 23	16, 23
0	Static CRTM Static Operating System	None	0 – 15
1	Dynamic Operating System	None	20
2	Dynamic Operating System	20, 21, 22	17, 18, 19, 20, 21, 22
3	An auxiliary level of a trusted component	None	17, 18, 19, 20
4	Dynamic CRTM	17, 18, 19, 20	17, 18

3.2 Static Locality

3.2.1 Concepts

This locality provides the root of trust for the components of the Host Platform as they are initialized from the Host Platform's reset. This is the default RTM for all components.

This space is intended to be used by all operations not within the Dynamic chain of trust or the Dynamic Operating System. This can occur before, during, or after the invocation of the Trusted Operating System (T/OS). Locality 0 transports a command to the TPM without any locality modification. PCR register modifications, uses or commands that require locality must not execute when presented using Locality 0. There must also be a new memory-mapped port defined for Locality 0.

As part of Locality 0 is Locality Legacy (i.e., using the legacy I/O port). This space is the same as Locality 0 from a PCR/TPM permission perspective, but is separate on the LPC bus. When any activeLocality bit is set, cycles on the legacy I/O port are aborted.

3.2.1.1 Initial TBB Control and Host Platform Reset

Upon Host Platform Reset, the CRTM MUST have control of the TBB. At this time, the localityModifier MUST be 0.

3.2.1.2 Static Core RTM (S-CRTM)

The Core Root of Trust for Measurement (CRTM) MUST be an immutable portion of the Host Platform's initialization code that executes upon a Host Platform Reset. The Host Platform's execution MUST begin at the CRTM upon any Host Platform Reset.

The trust in the Host Platform is based on this component. The trust in all measurements is based on the integrity of this component.

Currently, in a PC, there are at least two types of CRTM architectures:

1. CRTM is the BIOS Boot Block.

Start of informative comment

In this architecture the BIOS is composed of a BIOS Boot Block and a POST BIOS. Each of these are independent components and each can be updated independent of the other. In this architecture, the BIOS Boot Block is the CRTM while the POST BIOS is not, but is a measured component of the Chain of Trust.

End of informative comment

The manufacturer MUST control the update, modification, and maintenance of the BIOS Boot Block component, while either the manufacturer or a third-party supplier may update, modify, or maintain the POST BIOS component. If there are multiple execution points for the BIOS Boot Block, they must all be within the CRTM.

2. CRTM is the entire BIOS.

Start of informative comment

In this architecture, the BIOS is composed of a single atomic entity. The entire BIOS is updated, modified, or maintained as a single component. In this architecture, the entire BIOS is the CRTM.

End of informative comment

The manufacturer MUST control the update, modification, and maintenance of the entire BIOS.

3.2.1.3 Transferring Control

Prior to transferring control to another entity within Locality 0, an executing entity MUST measure the entity to which it will transfer control.

3.2.2 Integrity Collection and Reporting

Start of informative comment

The Static PCRs are divided into two primary sets. The first set is designated for the Host Platform's Pre-Operating System State (PCR[0-7]) and the other designated for the Host Platform's Static Operating System Present State (PCR[8-15]). The Static Pre-Operating System State's PCRs provide the Host Platform's initial chain of trust starting from Host Platform Reset. These establish chain of trust from the S-CRTM through the operating system's IPL Code. The definition of the Static Operating System Present State's PCRs are outside the scope of this specification and is the purview of either the specific operating system provider or a TCG operating system-specific specification, if one exists.

End of informative comment

1. The S-CRTM MUST be designed to perform integrity measurements of the Host Platform's Pre-Operating System State per the following sections.
2. This feature MAY be available on the Host Platform upon delivery to the Owner or MAY be made available to the Owner using a method provided by the Host Platform manufacturer. The Host Platform Certificate SHOULD indicate in which condition (i.e., pre-installed or updated) the Host Platform was delivered.

3. The Host Platform MAY provide a method for the Owner or Operator to disable these measurements.
4. Integrity measurements by the S-CRTM into PCR [0-7] MUST be done only while the Host Platform is in the Pre-Operating System State after starting from an S4 or S5 state; integrity measurements of the Pre-Operating System State MUST NOT be done to the PCRs allocated to the Operating System Present State. Any Host Platform configuration change that occurs after the Host Platform transfers control to the Operating System Present State, or while resuming from other power states, is the purview of the Operating System Present State and measurement of those events MUST be done to the PCRs allocated to the Operating System Present State.
5. If the TPM is deactivated, events (including EV_SEPARATOR) MUST NOT be logged into the event log.
6. If the capability to measure is either not installed or is disabled by the Host Platform Owner: Events other than EV_SEPARATOR MUST NOT be logged into the event log. However, the event EV_SEPARATOR MUST be measured into all Pre-Operating System State's PCRs as specified in Section 8.2.3. The event field for the EV_SEPARATOR event MUST be NULL.

3.2.3 PCR Usage

Start of informative comment

PCR[0:7] represent the Host Platform's static RTM; therefore, all the PCRs associated with Locality 0 are resettable only upon Host Platform Reset.

This section defines the PCR assignments used for boot time integrity metrics and the methodology for collecting the metrics. The first eight PCRs are defined for use within the Pre-Operating System State. (PCR[4] containing the measurement of the transition code between the Pre-Operating System State and the Operating System Present State.) Throughout the BIOS boot process a log of all executable code is created and extended into PCRs as described in Table 2.

Each time a PCR is extended, a log entry is made in the TCG Event Log. This allows a Challenger to see how the final PCR digests were built.

End of informative comment

Table 2 summarizes the defined PCR usage.

Table 2: PCR Usage

PCR Index	PCR Usage
0	CRTM, BIOS, and Host Platform Extensions
1	Host Platform Configuration
2	Option ROM Code
3	Option ROM Configuration and Data
4	IPL Code (usually the MBR)
5	IPL Code Configuration and Data (for use by the IPL Code)
6	State Transition and Wake Events
7	Host Platform Manufacturer Control
8-15	Defined for use by the Static Operating System. Host Platform

3.2.3.1 PCR[0] – CRTM, POST BIOS, and Embedded Option ROMs

Start of informative comment

The CRTM may measure itself to PCR[0] and must measure to PCR[0] any portion of the POST BIOS, including manufacturer-controlled embedded Option ROMs, Host Platform firmware, etc., that are provided as part of the motherboard. Only executable code is logged. Configuration data such as ESCD should not be measured as part of this PCR.

All these components and any update to them are under the control of the manufacturer or its agent.

If, for any reason, a measurement cannot be made to this PCR, none of the following PCR values can be trusted and, therefore, are outside the chain of trust. It is, therefore, necessary to invalidate all Host Platform PCRs.

Because the measurement of the POST is typically done within a resource constrained environment (i.e., the S-CRTM, likely the BIOS Boot Block), the event log corresponding to the extend event cannot be created at the time the TPM_Extend is performed. It is acceptable to have the POST generate the corresponding event log entry, reconstructing it from information (i.e., the value that was extended) passed to it from the BIOS Boot Block. If this procedure is used, the POST must be sure the entries within the event log are properly sequenced, i.e., represent same order as the sequence of TPM_Extend operations.

Measurement of Non-host Platforms

If a Non-host Platform cannot be reliably detected by the BIOS and measured into PCR[0], the existence of that Non-host Platform should be indicated in the Host Platform Certificate.

Design Consideration and Distinctions Between PCR[0], PCR[2], and PCR[4]

PCR[0] typically represents a more consistent view of the Host Platform between boot cycles. This allows Attestation and Sealed Storage policies to be defined as those using the less changeable components of the transitive trust chain. This PCR contains the components provided by the Host Platform manufacturer; therefore, the verifier or the entity performing the seal operation can choose to seal to only those components provided and updated by the Host Platform's manufacturer. This is also the reason embedded Option ROM binaries are measured into PCR[0], as well, thus providing this same consistent view of the platform regardless of user selected options.

PCR[2] is intended to represent a more "user" configurable environment where the user has the ability to alter the components that are measured into PCR[2]. This is typically done by adding adapter cards, etc., into "user" accessible PCI or other slots.

PCR[4] is intended to represent the entity that manages the transition between the platform's Pre-Operating System Start and the Operating System Present State. This PCR, along with PCR[5], identifies the initial operating system loader.

End of informative comment

Entities that MUST be Measured:

1. The S-CRTM's version identifier.
2. All Host Platform firmware physically bound to the motherboard which is executed by the Host Platform's CPU(s) and is part of the Host Platform's Transitive Trust Chain. This includes, but not necessarily limited to, the following:
    a. POST code
    b. Embedded SMM code and the code that sets it up
3. ACPI Flash data prior to any modifications.
4. BIS code (excluding the BIS Certificate).
5. Manufacturer-controlled embedded Option ROMs as a binary image.
    a. These are embedded Option ROMs. Their release and update is controlled by the Host Platform manufacturer.
6. The BIOS MUST attempt to detect and measure the presence of any Non-host Platform. If the BIOS detects the presence of a Non-host Platform, it MUST measure relevant information about its presence such as type, version, etc., into PCR[0] using event EV_NONHOST_INFO.

Entities that MAY be Measured:

1. The S-CRTM itself.
2. Any other code or information that is relevant to the S-CRTM, POST BIOS, or Host Platform Extensions.
3. Components within Non-host Platforms (e.g., firmware not intended to be executed by the Host Platform's CPU) that are not part of the Host Platform's Transitive Trust Chain but may affect the trust of the Host Platform or System.

Entities that SHOULD NOT be Measured:

1. Changes to the user setup configuration SHOULD NOT be measured into PCR[0].

Error condition:

If the measurement of the CRTM, POST BIOS, and embedded Option ROMs cannot be made, the CRTM MUST be capped by measuring the value 01h to each PCR[0-7]. If this cannot occur, the Host Platform SHOULD take any necessary action to notify the Host Platform's administrator, user, and operator along with transitioning into a "fail-safe" mode.

Method for Measurement for a Compound BIOS:

The CRTM performs these measurements as follows:

1. Log the CRTM's version identifier.
2. Measure the code to which the CRTM is transferring control.

    The POST BIOS may need to reconstruct events that could not be recorded in the event log due to the unavailability of memory. If it does so, it places this information into the Event Log and MUST NOT extend PCR[0] with this reconstructed information.
3. The remaining measurements MAY be performed in any order.

Method for Measurement for an Integrated BIOS:

The CRTM performs these measurements as follows:

1. Log the S-CRTM's version identifier.
2. The S-CRTM measures the remainder of the BIOS firmware.

3.2.3.2 PCR[1] – Host Platform Configuration

Start of informative comment

Information about the configuration of the motherboard including hardware components and how they are configured is measured to PCR[1].

The BIS Certificate may contain information that is privacy sensitive; thus exclusion of the BIS Certificate is allowed.

The method and policy of disabling measurements is manufacturer specified.

End of informative comment

Entities that MUST be Measured:

The following entities MUST always be measured. These MUST NOT be disabled:

1. If the BIOS loads a CPU microcode update, it is measured.
2. The Host Platform Configuration event EV_PLATFORM_CONFIG_FLAGS MUST be measured.

Entities that MAY be Measured

The following entities MAY be measured. Even if measurement of these entities is provided by the BIOS, the BIOS MAY allow the Owner to Disable measurement of them.

1. BIS Certificate.
2. POST BIOS-Based ROM strings.
3. Table of Devices.

   This allows the BIOS to measure the list of devices attached to the Host Platform. Examples of this include PCI devices, onboard Video adapters, etc. Because of the wide variance of Host Platform architectures, the actual format of the data providing this information is left to the Host Platform manufacturer. It is left to the challenger to discover the format of this data. It could, for example, reference the Host Platform Certificate, and then contact the Host Platform manufacturer to obtain this information. This data is encapsulated within the structure defined in Section 10.4.2.3.13. The BIOS MAY create multiple entries of this event or MAY choose to encapsulate all the data into a single entry.

   a. All PCI devices: both embedded and add-in cards.
   b. Onboard Video Adapter.
   c. Others as defined by the Host Platform manufacturer.
4. ESCD, CMOS, and other NVRAM data.
5. SMBIOS structures.

Entities that MUST NOT be Measured

1. Values and registers that are automatically updated (e.g., clocks).
2. System unique information such as asset, serial numbers, etc.

Passwords

Method for Measurement:

The BIOS performs these measurements as follows:

1. The entities specified in this PCR MAY be measured in any order deemed appropriate by the implementer. Where possible, these measurements SHOULD occur prior to measuring Option ROMs.

3.2.3.3 PCR[2] – Option ROM Code

Start of informative comment

Option ROMs contained on Non-host Platform adapters are measured by the BIOS to PCR[2]. There may be two portions of Option ROMs: visible and hidden. Each is measured and logged to PCR[2].

Visible Portion

The portion of the Option ROM that is visible to the BIOS MUST be measured by the BIOS.

Hidden Option ROM Code

Some Option ROMs may use paging or other techniques to load and execute code that was not visible to the BIOS when measuring the visible portion of the Option ROM. It is the responsibility of the Option ROM to measure this code prior to executing any portion of that hidden Option ROM code.

End of informative comment

The BIOS MUST measure the visible portion of the Option ROM into PCR[2] prior to executing it. In all cases, Option ROM code that is executed MUST be measured even if the binary representing the code was already measured into PCR[0].

Any application that modifies the Option ROM code MUST measure the new code into PCR[2] or cause a Host Platform Reset.

Entities to be Measured:

1. The portion of the Option ROM that is visible to the BIOS.
2. The portion of the Option ROM that is not visible to the BIOS is measured by the Option ROM.
3. Non-manufacturer-controlled embedded Option ROMs

   These are embedded Option ROMs that are physically contained on the motherboard (as opposed to an add-in card), but the release and control of any update is not controlled by the (motherboard) manufacturer.

Method for Measurement:

The BIOS performs these measurements for each discovered Option ROM as follows:

1. Measure the event OptionROMExecute for each Option ROM.
2. The entities specified in this PCR MAY be measured in any order deemed appropriate by the implementer.
3. Repeat until all Option ROMs are measured and executed.

Option ROMs perform these measurements as follows when they execute:

1. Measure the event "Un-hiding Option ROM Code" when un-hiding Option ROM code.
2. Measure the "hidden" Option ROM Code prior to executing it.

3.2.3.4 PCR[3] – Option ROM Configuration and Data

Start of informative comment

As Option ROMs execute, they may have configuration and other data relevant to the trusted properties of the Host Platform. Option ROMs perform this measurement.

An example of information measured into PCR[3] is a SCSI controller's configuration of its hard disks; e.g., RAID type, drive assignments, etc.

End of informative comment

Any application that modifies the Option ROM configuration MUST measure the new configuration into PCR[3] or cause a Host Platform Reset.

Entities to be Measured:

1. Configuration data specific to the Option ROM or the adapter that hosts the Option ROM.
2. Other data, including comments, specific to the Option ROM or the adapter that hosts the Option ROM.

Method for Measurement:

The Option ROM or application performs these measurements as follows:

1. Measures the event OptionROMConfigConfiguration.
2. Measure any of the above in any order while executing.

3.2.3.5 PCR[4] – IPL

Start of informative comment

If IPL Code returns control back to the BIOS, each subsequent execution of IPL Code must be separately measured.

End of informative comment

Entities to be Measured:

1. Each IPL that is attempted and executed.
2. Additional code that is loaded by the IPL Code.

Entities to Exclude:

1. Portions of the IPL Image pertaining to the specific configuration of the Host Platform (e.g., disk geometry in the MBR).

Method for Measurement:

This section provides only a short overview of the actions required. See Section 8.2.3 for specific details.

The BIOS performs these steps as follows:

1. Measure EV_ACTION with the relevant event.
2. Measure the IPL Code.
3. If control returns to the BIOS, measure the returning event.
4. Go to Step 1.

A complete description of the method for measuring IPL Code is found in Section 8.

3.2.3.6 PCR[5] – IPL Configuration and Data

Start of informative comment

The IPL Code may have configuration or other data that is relevant to the trusted properties of the Host Platform. An example of this is IPL Code that allows the selection of alternate boot partitions. In this example, the partition selection information would be logged to this PCR by the IPL Code.

Information measured into this PCR by the BIOS is static information embedded within the IPL Code such as the disk geometry within the MBR.

End of informative comment

Entities to be Measured:

1. All relevant IPL configuration data.
2. Static data contained within the IPL Code (e.g., disk geometry).

Method for Measurement:

1. The IPL Code measures all relevant IPL configuration data per its defined events.
2. The BIOS measures the static data such as disk geometry.

3.2.3.7 PCR[6] – State Transition

Start of informative comment

Resuming from S5 (i.e., initial Host Platform bootstrap) and S4 (resume from hibernation) are on-time events from the perspective of the RTM and the chain of trust. All other resume conditions retain the initial chain of trust. In addition, other wake events tend to be very time sensitive. Measurements can cause a significant increase in some state transition times when considering them as a percentage. As the operating system actively participates in state transitions other than resume from S5 / S4, it is best to leave measuring these events to the operating system. The EV_SEPARATOR that will be measured into this PCR prior to turning control of the Host Platform to the operating system will provide the delimiter between the wake event measured into this PCR by the Pre-Operating System

State's components and those that have been measured into this PCR by the Operating System Present State's components. Note: This PCR is used by both the Pre-Operating Systme State and Operating System Present State's components.

End of informative comment

Entities to be Measured:

1. Resuming from S5 or S4

Entities that MUST NOT be Measured

1. Resuming from S3 through S1

Method for Measurement:

1. Measure the EV_ACTION with Action Index 10, "Wake Event n", where the value of "n" is determined by the state from which the Host Platform is resuming.

3.2.3.8 PCR[7] – Host Platform Manufacturer Control

Start of informative comment

This PCR is reserved for use by the Host Platform manufacturer. This allows the Host Platform manufacturer-specific applications that operate during the Pre-Operating System State, to use this PCR. The use of this may not be common across different Host Platform manufactures and even across different Host Platform models within the same Host Platform manufacturer. Regardless of its use, this PCR contains an EV_SEPARATOR as specified in Section 8.2.3.

End of informative comment

1. The Host Platform manufacturer MAY define the purpose of this PCR.
2. User applications MUST NOT use this PCR for Sealing or Attestation.

3.2.3.9 PCR[16] – Debug

Start of informative comment

This PCR is resettable from any locality and is for use by any entity on the Host Platform. It is intended, by convention, to be used as a debug PCR. Components should use this PCR for debugging purposes only (e.g., software development of components utilizing the PCR features of the TPM, e.g., TPM_Seal). Applications targeted for user, final, or production environments should not use this PCR in their final release.

End of informative comment

1. Any component on the Host Platform MAY use and reset PCR[16] at any time.
2. User applications MUST NOT use this PCR for Sealing or Attestation.

3.2.3.10 PCR[23] – Application Support

Start of informative comment

This PCR is resettable from any locality. It is to be used by the operating system or its applications.

End of informative comment

1. The operating system or its applications define the purpose of this PCR and MAY reset and use it at any time.

3.2.4 Localities Assigned to the Dynamic RTM and Transitive Trust Chain

3.2.4.1 Concepts

Start of informative comment

The Static Operating System defines the usage of the Locality 0 PCRs (i.e., PCR[8-15] and PCR[23]). The Host Platform manufacturer and Dynamic Operating System defines the usage of Localities 1-4 (i.e., PCR[17-22]). PCR[16] is reserved for debugging purposes. Its usage is, therefore, undefined. The usage of Localities 1-4 is, therefore, beyond the scope of this specification. However, the Host Platform is responsible for maintaining protections of the PCRs based on their associated locality. The normative reference of the relationship between localities and PCRs and their relative attributes is specified in the *PC Client TPM Interface Specification*. However, for convenience, it is duplicated here and represented from a perspective more useful to the operating system and application developer.

Locality	Associated PCR	Usage	Rest Capability	Extend Capability
4	17[3]	Associated with the D-CRTM	PCR[17-20].	PCR[17-18]
3	18	Host Platform defined	None	PCR[17-20]
2	19	Trusted Operating System	PCR[20-22]	PCR[17-22]
1	20	Used by Trusted Operating System	None	PCR[20]
TOS Controlled	21	Used by Trusted Operating System	N/A	N/A
TOS Controlled	22	Used by Trusted Operating System	N/A	N/A

End of informative comment

3.2.5 Dynamic Core RTM (D-CRTM)

This MUST be an immutable portion of the Host Platform but is not required to begin at the Host Platform Reset. The location and method of executing this is Host Platform implementation dependent but MUST be a Trusted Process. The Host Platform Certificate MAY state the method for invoking the D-CRTM.

While the D-CRTM executes after, and in some respects within, the S-CRTM, the value of the D-CRTM's transitive trust chain does not depend on the S-CRTM's transitive trust chain.

[3] The PCRs associated with the Static RTM and the debug PCR are ignored in this case for simplification.

3.2.6 Locality 1- 3

3.2.6.1 Integrity Collection and Reporting

The method of collecting and reporting these PCRs is beyond the scope of this specification and is the purview of the Host Platform's hardware that supports the D-RTM.

3.2.6.2 PCR Usage

The usage of these PCRs is beyond the scope of this specification and is the purview of the Dynamic Operating System.

3.2.6.3 Protection

The Host Platform MUST provide protections to enforce the locality messages from the source of the commands to the TPM.

3.2.7 Locality 4

3.2.7.1 Concepts

Start of informative comment

(Dynamic Core Root of Trust [Trusted hardware locality]) – indicates the command was generated by D-CRTM or trusted hardware on the Host Platform. The Locality 4 modifier CANNOT be generated by any untrusted software including the T/OS. Locality 4 transports the command to the TPM and uses mechanisms that cannot be misinterpreted as being any other locality modifier. For the Locality 4 PCR, the TPM_PCR_RESET command must require the Locality 4 modifier in order to be executed. The TPM_HASH_* LPC commands are done with Locality 4, since it is hardware generated.

End of informative comment

The assertion of Locality 4 MUST only be performed by the D-CRTM.

3.2.7.2 Integrity Collection and Reporting

How this is done is Host Platform implementation specific.

3.2.7.3 PCR Usage

The PCR associated with this locality is PCR[17]. This locality MUST measure the first component executed after the D-CRTM and MAY measure other components as determined by Host Platform's implementation. This is analogous to PCR[0] in the S-RTM.

3.2.7.4 Protection

The Host Platform MUST provide protections to enforce that Locality 4 commands can only originate from the D-CRTM.

3.2.8 Operating System Resettable PCR

3.2.8.1 Concepts

Start of informative comment

These PCRs are under control of the Dynamic Operating System. The usage is determined by the Dynamic Operating System just as the usage of PCR[8-15].

End of informative comment

3.2.8.2 Integrity Collection and Reporting

What components are measured and reported are defined by the Dynamic Operating System.

3.2.8.3 PCR Usage

PCR[21:22] can be reset and extended to by the Dynamic Operating System.

4. Non-volatile Storage

Start of informative comment

Non-volatile Storage (NV RAM) is made available by the TPM for use by various processes on the Host Platform. In keeping with the general philosophy of TCG architecture, there is a limited amount of resources that are required to be provided by the TPM.

End of informative comment

4.1 NV RAM Size and Allocation

Start of informative comment

The TIS specifies a minimum amount of NV Storage space the TPM must provide. However, this value is based on the anticipated usages of this area at the time the specifications are written. It is expected that the operating system and its applications will require at least 512 bytes to be available. This means the Host Platform manufacturer must calculate its usage of this area and use a TPM that provides sufficient NV Storage space to accommodate the Host Platform's usage while leaving 512 bytes of unallocated NV Storage space for use by the operating system and its applications.

End of informative comment

1. After all the NV Storage requirements of the Host Platform manufacturer are satisfied, the Host Platform MUST provide to the Owner at least 512 bytes for use by Localities 2 and 1 total.

CAVEAT: The above calculation and resulting value represents an estimate prior to any released implementation. TPM and Host Platform manufacturers are encouraged to consult each other regarding implementation-specific requirements. The Host Platform manufacturer is also encouraged to take into consideration the uses of the Host Platform. The value 512 is based on "anticipated" use but not all environments are equal and some may require more.

4.2 NV Storage Indexes

4.2.1 TCG Main Specification Reserved Indexes

Start of informative comment

It is important, and in most cases necessary, for Host Platform software to obtain Host Platform-related TCG certificates. These certificates may, but not necessarily, be created by the TPM and Host Platform manufacturer. There is no required mechanism for distribution of these TCG certificates. However, providing a "standard" and agreed upon location to store them provides a convenience for both the Host Platform manufacturer and the software as both have a well known location to store (on the part of the Host Platform manufacturer) and retrieve (on the part of the software). These certificates contain no security-sensitive information (i.e., there is compromise of the security properties of the TPM or the Host Platform); however, they do contain privacy-sensitive information. Unauthorized exposure may result in loss of privacy for the Owner or users of the Host Platform.

Because the Owner is responsible for the privacy of the Host Platform and these certificates are for use only by the Owner, only the TPM Owner should have the ability to obtain them.

One of the NV Storage attributes is TPM_NV_PER_OWNERREAD which means only the "current" Owner has authorization to read the specified NV Storage area. However, this area is cleared upon the deletion of the first Owner after this area is populated. Therefore, it is the responsibility of the Owner giving up ownership of the Host Platform to transfer, if appropriate, the certificates to the new Owner.

End of informative comment

In the *TPM Main Specification, Part 2 Structures*, Section 19.1.2, the following indexes are defined and reserved:

1. TPM_NV_INDEX_EKCert
2. TPM_NV_INDEX_TPM_CC
3. TPM_NV_INDEX_PlatformCert
4. TPM_NV_INDEX_Platform_CC

If any of these indexes are used, the TPM and the Host Platform manufacturer MUST set the data to the indicated certificate using the format specified in the following section. Subsequent Owners SHOULD retain these values or replace them with correspondingly new values of the same type.

For any of the above indexes that are used, the Host Platform manufacturer MUST set the TPM_NV_PER_OWNERREAD bit in the TPM_NV_ATTRIBUTES->attributes to TRUE.

4.2.2 PC Client Host Platform Reserved NV RAM Indexes

Start of informative comment

Some environments within the PC Client Host Platform may require exclusive use of some NV RAM area. To avoid collision with other entities within the Host Platform, the indexes listed in Table 3 are reserved for the indicated entity.

End of informative comment

The processes within the Host Platform MUST reserve the following ranges for use by the indicated entity. The meaning, use, and value of the data referenced by these indexes is implementation-dependent and neither specified nor enforced. The Reserved index values MUST NOT be used.

Table 3: NV RAM Index Values

Value Range	Entity Name
0x0000F500 - 0x0000FFFF	Reserved
0x0000F400 - 0x0000F4FF	Locality 4
0x0000F300 - 0x0000F3FF	Locality 3
0x0000F200 - 0x0000F2FF	Locality 2
0x0000F100 - 0x0000F1FF	Locality 1
0x0000F000 - 0x0000F0FF	Preallocated

0x00000002 - 0x0000EFFF	Locality 0[4] or General Purpose

For any of the above indexes that are used, the Host Platform manufacturer MUST set the TPM_NV_PER_OWNERREAD bit in the TPM_NV_ATTRIBUTES->attributes to TRUE. Any subsequent Owner of the Host Platform SHOULD retain this attribute value.

[4] The DIR index is reserved for backward compatibility with the *TCG Main Specification, Version 1.,1* and is located at an index with a size requirement as specified in *TCG Main, Part 2 Structures, Version 1.2*, Section 19.1.1.

5. 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 "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 Specification* to specify the routing to the specific device. In general, this is done by mapping the data sent in the TPM_NV_WriteValue*[5] 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 that time, this area returns to undefined and must again be defined before use. The reason for this is the new TPM Owner may have different security and privacy requirements for this GPIO.

The range reserved for GPIO is one that 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).

Defined Uses:

Currently the only defined usage of the GPIO is for use by the GPIO-Express-00 pin, which allows software to control an 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. The *PC Client TPM Interface Specification* maps the value of the least significant bit of TPM_NV_INDEX_GPIO_00 data to the GPIO-Express-00 pin. This bit will be called the GPIO-Express-00 bit for documentation purposes.

Reviewing the *PCI Express Trusted Configuration Space ECR*, it is important to note the negative logic used. It is better for both security and privacy that this signal be inactive by default and remains inactive until specifically driven active. Circuitry is easier and more cost effective when designing the default to be high and having the components drive the

[5] For simplicity in documentation, references to the TPM_NV_WriteValue* command mean either the TPM_NV_WriteValue or the TPM_NV_WriteValueAuth command, and references to the TPM_NV_ReadValue* command mean either the TPM_NV_ReadValue or the TPM_NV_ReadValueAuth command.

signal low only when specifically driven to do so. This ECR states that the TCS is enabled when the TCS_EN pin is active but that it is active when the signal is low.

Note to Implementers:

Careful examination of the TPM_NV_ATTRIBUTES demonstrates that if none of the read 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 *de*activates the TCS while a write of a '0' to the GPIO-Express-00 bit *activates* the TCS.

[Editor's Note: The above is a copy of text 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.]

Because the purpose of GPIO is to provide an interface between the TPM's command interface and specific pin(s) on the TPM, Host Platform manufacturers and software developers are encouraged to read the relevant sections of the PC Client TPM Interface Specification to gain a full understanding of its use. Of particular interest is the detailed description of the allocation and methods for associating the particular NV Storage area with the GPIO and the methods for performing the actual reads and writes.

Note: Use and implementation of the currently defined set of GPIOs is optional. However, if any of the defined set of GPIOs is implemented, it must be implemented in the manner prescribed.

Security Implications

The states of the TPM's GPIO signals are not assured until after TPM_Startup completes. This puts the privacy and even the security of some aspects of the Host Platform under some control of the S-CRTM or any component executing prior to TPM_Startup.

End of informative comment

1. Implementation of the GPIO-Express-00 bit and pin is optional. If it is implemented, it MUST be implemented as defined in this section.

The following text applies if, and only if, the Host Platform is implemented with a chipset that supports the PCI Express TCS_EN pin.

Note: Detailed descriptions of the mechanisms used for allocation, writing, and reading (e.g., bit position assignments, etc.) are provided in the *PC Client TPM Interface Specification*. Some of the descriptions used here are generalizations of those more detailed descriptions.

2. The GPIO-Express-00 pin MUST be connected to the PCI Express TCS_EN pin of the PCI Express Root Complex as described in *PCI Express Trusted Configuration Space ECR*.
3. The Host Platform MUST be designed such that only the GPIO-Express-00 pin is able to drive the TCS_EN pin.
4. The TCS_EN signal is active low; i.e., when the TCS_EN is high, the PCI Express Trusted Configuration Space is disabled. When the TCS_EN is low, the PCI Express Trusted Configuration Space is enabled.
5. The Host Platform MUST be designed such that the default state of the TCS_EN pin is inactive.

6. The Host Platform MUST be designed such that entering, exiting, and during any sleep state (i.e., any state other than S0), the TCS_EN pin defaults to inactive whenever the TCS could be utilized.
7. The GPIO-Express-00 bit SHALL be the least significant bit of TPM_NV_INDEX_GPIO_00.
8. The connection from the TPM's GPIO-Express-00 bit to the GPIO-Express-00 pin MUST be such that:
    a. Writing a "1" to the GPIO-Express-00 bit results in holding the GPIO-Express-00 pin to a high state.
    b. Writing a "0" to the GPIO-Express-00 bit results in holding the GPIO-Express-00 pin to a low state.
    c. Reading the GPIO-Express-00 bit returns the current state of the GPIO-Express-00 pin.
9. The Host Platform MAY implement other GPIOs but MUST NOT use any NV Storage areas marked as "Reserved" in the *PC Client TPM Interface Specification*.

6. Maintenance

Start of informative comment

Maintenance for the *TCG PC Specific Implementation Specification* refers to the processes surrounding upgrade or replacement of the system BIOS ROM / Flash. All requirements for TPM maintenance are manufacturer defined.

As noted in Section 1.5.3, security requirements are not defined in this specification. The security requirements include the optional maintenance features defined in this section. There is a separate package in the Protection Provide targeted to maintenance.

End of informative comment

Implementation of Maintenance is optional. If it is implemented, it MUST be implemented as defined in this section.

6.1 BIOS Recovery Mode

Start of informative comment

This is a failure-recovery mode of the BIOS that is invoked by the BIOS Boot Block typically when the main BIOS is corrupt. The BIOS Recovery mode will perform a minimal initialization of the system and then attempt to boot a recovery program from some type of external media, e.g., from floppy disk.

A couple of attack scenarios have been identified due to the “BIOS Recovery” feature implemented in many BIOSes. A method to counter these attacks could implement:

1. A hardware signal (e.g., GPIO or similar) to “disable” the TPM (until the next TPM_Init)
2. Modifications to CRTM recovery code which would extend the value 1 to the Pre-Operating System State’s PCRs
3. Disable the TPM by calling TPM_Startup (stTrpe=TPM_ST_DISABLE)

End of informative comment

It MUST NOT be possible for a BIOS Recovery Mode to allow impersonation of another valid boot state. This applies to the values in the Pre-Operating System State’s PCRs. Upon completion, the BIOS Recovery Code MUST cause a Host Platform Reset.

6.2 Flash Maintenance

Start of informative comment

There are two scenarios: A Manufacturer Approved Environment (MAE) and a Non-MAE. The MAE may update any portion of the BIOS while the Non-MAE may not update the CRTM.

End of informative comment

6.2.1 Manufacturer Approved Environment (MAE)

Start of informative comment

This environment provides the same strength of protections provided by the original TBB manufacturing process. Using a utility that is approved by the manufacturer of the Host Platform is equivalent to being done within an MAE.

End of informative comment

The CRTM MAY be updated while within an MAE.

6.2.2 Non-Manufacturer Approved Environment (NMAE)

Start of informative comment

This is an environment that does not have the equivalent strength of the original TBB manufacturing process. Using a utility that is not approved by the manufacturer of the Host Platform is considered to be a NMAE.

End of informative comment

The CRTM MAY NOT be updated while within an NMAE.

7. TCG Certificates and Verification of a Platform

While this specification does not mandate TCG certificates be issued, any TCG certificate MUST be represented as defined by the TCG Infrastructure Working Group. The statements made in this specification regarding the contents of the certificates are general in nature and are intended to provide guidance to the issuers of those certificates. Credential issuers are directed to search for TCG Specifications dealing specifically with mapping the aspects of the Platform, including those specifically mentioned in this specification, into attributes of TCG Certificates.

Informative comment

The methods for verifying a platform's trust properties are beyond the scope of this specification. However, some guidance may prove useful in understanding some of the concepts that have lead to the development of this and other TCG Specifications. The verifier, which could be an end user on a non-networked platform or a sophisticated component within a policy-driven enterprise, uses one or more criteria to determine the trustworthiness of the platform. One of these criteria is the verifier's knowledge and trust in the source of the platform's trust components or the RTM. This can be done using methods as simple as trusting the platform manufacturer. Further, this might also involve the verifier examining the platform's design and features. Assuming the verifier trusts the platform manufacturers, how does the verifier know the platform that is being challenged is actually from that platform manufacturer? One method is simply to trust the delivery mechanism. Another method is to verify and examine a Platform Certificate issued by the Platform manufacturer.

Another source of information, if one is produced and is related to the Platform Credential, is the Security Target. This document states the various security properties of the platform. Further trust may be obtained if the platform has been evaluated by a trusted third-party such as an evaluation lab.

End of informative comment

7.1 Host Platform Certificate

Distribution is manufacturer controlled.

7.2 Host Platform Conformance Certificate

Distribution is manufacturer controlled.

7.3 Validation of Objects Using a Validation Certificate

Informative comment

Validation Certificates provide certified and trusted information about expected measurement value of an object. An example of an object here would be an Option ROM where the Option ROM contains the Validation Certificate or a reference to it. If a Validation Certificate is available, it may not be necessary to measure the contents of the Option ROM, rather the BIOS may compare the measured but not extended contents of the Option ROM to the value contained within the Validation Certificate. The result of this comparison is then measured into the PCR.

End of informative comment

7.3.1 Method of Verification

Verification of an object against the hash value within the Validation Certificate is not required. If this verification is performed, the hash within the Validation Certificate must include the entire Validation Certificate Header excluding the Validation Certificate itself.

7.3.2 Validation Certificate Header

If present, the Validation Certificate MUST be contained within the Option ROM header as specified in Table 4 according to the *Plug and Play BIOS Specification*.

Table 4: Validation Certificate Header

Offset	Size	Value	Description
0h	DWORD	41h, 50h, 43h, 54h	This is the byte sequence 41h, 50h, 43h, 54h which is the ASCII string 'TCPA' for compatibility with the original specification.
04h	BYTE	01h	Structure Revision
05h	BYTE	Varies	Length (in 16-byte increments)
06h	WORD	Varies	Offset of next Header (0000 if none)
	BYTE	Varies	Number of segments. Value of 0 indicates entire visible portion of Option ROM excluding the Validation Certificate.
	WORD	Varies	Offset to first segment included in Validation Certificate hash
	WORD	Varies	Length-1 of first segment included in Validation Certificate hash
	…	…	Repeat for number of segments
	…	…	
??h	BYTE	0FFh	Reserved
??h	BYTE	Varies	Checksum of this entire header as specified in the *Plug and Play BIOS Specification*
??h	Varies	Varies	Validation Certificate

7.4 Storing Certificate in TPM NV Storage

Informative comment

There are many methods for distributing TCG certificates; for example, with the platform's distribution CD, within a partition on the platform's hard disk, on the TPM or platform manufacturers web site, etc. One method, described in this section, is to store them within the NV Storage area of the TPM. Storing and distributing the certificates using this method should be done with caution. There is no requirement to use this method for distributing certificates.

As with most data contained within the TPM's NV Storage area, this data is opaque to the TPM and its operations. It has meaning and context only to the applications writing and reading it.

Protections must be put in place to prevent the exposure of system unique information to unauthorized entities. Access to this storage element MUST be restricted for privacy reasons. A typical use would be to store this in an NV store location that requires Owner authorization for read.

End of informative comment

1. TPM and Host Platform manufacturers MAY use the methods described in this section to distribute TCG or other Host Platform certificates.
2. Access to Host Platform unique information MUST NOT be available to unauthorized entities and SHOULD NOT be available to "public".

7.4.1 Certificate Structure Tags

Informative comment

These definitions are, in some ways, parallel to the TPM_STRUCTURE_TAG defined in the TPM Specification. However, as the structures using them defined with this specification are never processed by the TPM itself, their purview is within this platform specification.

End of informative comment

7.4.1.1 Helper Definitions

Table 5: Helper Definitions

Name	Value	Description
TCG_PCCLIENT_STRUCTURE_TAG	UINT16	Identifies size of the tag

7.4.1.2 Structure Tag Definitions

These values are parsed into two fields. The upper nibble defines the purview as belonging to a TCG platform specific specification. The lower 3 nibbles define the value. The software is expected to understand which platform class this structure belongs to properly parse it.

Table 6: Structure Tag Definitions

Name	Value	Description
TCG_TAG_PCCLIENT_STORED_CERT	1001h	Defines a stored Certificate
TCG_TAG_PCCLIENT_FULL_CERT	1002h	Defines a full Certificate
TCG_TAG_PCCLIENT_PART_SMALL_CERT	1003h	Defines a partial small Certificate

7.4.2 TPM_CERT_TYPE

Informative comment

The type of certificate contained within the TCG_PCCLIENT_STORED_CERT.cert field.

End of informative comment

Table 7: Type of Certificate Representation

Name	Value	Description
TCG_FULL_CERT	0	The cert field contains a full certificate. (Can determine type of certificate by looking at its contents).
TCG_PARTIAL_SMALL_CERT	1	The storage element includes only the signature element of the certificate; the remaining portions of the certificate must be built from information available from the TPM, Host Platform, and/or local or remote storage.

7.4.3 TCG_CERT_FLAGS

Informative comment

This flag information describes the end certificate structure and storage organization.

End of informative comment

There are no defined flags. The value MUST be 0.

7.4.4 TCG_PCCLIENT_STORED_CERT

Informative comment

At the storage location is an instance of this structure, which provides the system with a mechanism to convert the storage space to an actual certificate.

End of informative comment

Definition

```
typedef struct tdTCG_PCCLIENT_STORED_CERT {
   TCG_PCCLIENT_STRUCTURE_TAG tag;       // 2 bytes
   BYTE                       certType;  // 1 byte
   UINT16                     certSize;  // 2 bytes
   BYTE[]                     cert;      //
} TCG_PCCLIENT_STORED_CERT ;             // Minimum total 5 bytes
```

Table 8: TCG_PCCLIENT_STORED_CERT Structure

Type	Name	Description
TPM_STRUCTURE_TAG	Tag	MUST be TCG_TAG_PCCLIENT_STORED_CERT
BYTE	certType	An element from the TPM_CERT_TYPE table
UINT16	certSize	The size of the certificate structure
BYTE[ ]	Cert	The certificate itself. If certType is TPM_SIG_CERT, this is a TPM_SMALL_CERT structure.

7.4.5 TCG_FULL_CERT

Informative comment

The cert field contains the entire certificate. The nature and type of the certificate is to be determined by its content.

End of informative comment

Definition

```
typedef struct tdTCG_FULL_CERT {
   TCG_PCCLIENT_STRUCTURE_TAG tag;     // 2 bytes
   BYTE[]                     cert     // Entire certificate
} TCG_FULL_CERT;                       // Minimum total 2 bytes
```

Table 9: TCG_FULL_CERT Structure

Type	Name	Description
TPM_STRUCTURE_TAG	tag	MUST be TCG_TAG_PCCLIENT_FULL_CERT.
BYTE	cert	The entire certificate.

7.4.6 TCG_PARTIAL_SMALL_CERT

Informative comment

While storing full certificates might be convenient to an application, the limited availability of NV Storage within the TPM makes this option difficult or even infeasible in most situations. For this reason, the structure defined in this section provides optimizations allowing only a small portion of the certificate to the stored within the limited NV Storage area. The *TPM Specification* contains reserved index values for the TPM and Host Platform Certificate.

It is assumed that much of the information contained within the certificate is actually duplicated on the platform or can be either obtained or derived from information available to a utility. The information stored within the NV Storage area, therefore, can be reduced to only that which is unique to that certificate that cannot be derived. For example, for a platform certificate, the platform's manufacturer and model is often available from other sources within the platform. There is no need to duplicate this information in the NV Storage area of the TPM. A utility, which can be provided by the TPM manufacturer, platform manufacturer, or both, can read this information and begin the process of constructing a certificate from information such as this. However, information such as the signature of the certificate cannot be derived. This later type of information is a good candidate for storing in the NV Storage area of the TPM.

It is, therefore, expected that utilities will be provided by the TPM or platform manufacturer to read the information from the TPM or platform or a URL, read the NV Storage area, and from these sources construct the certificate. The utility may even be able to validate the certificate using the signature obtained from the signature portion of this structure.

As currently implemented, the "partial, small" version of this structure is at least 269 bytes long for a single certificate using a 2048-bit key. Actual usage in an NV Storage area will require more total bytes of NV Storage in the TPM, depending on the implementation. A reasonable estimate is that the amount of physical NV memory required would be between 300 and 330 bytes (10% - 20% overhead).

It is expected that generation of the actual certificate will require at least some of the following additional information:

1. Serial and model numbers of the Host Platform. Obtained by the user in some way (programmatically, visually, etc.).

2. Endorsement Public Key. Obtained from the TPM in the usual way (requires the cooperation of the Owner).
3. Manufacturer of the TPM and Host Platform. TPM manufacturer can be obtained with getCapability; Host Platform manufacturer must be obtained by the user in some way (visual, programmatically, etc.).
4. An array of non-unique certificate templates from which to choose. Expected location for this array would be a distribution CD or other disk, a web site, or other location. The model number and manufacturer would be used to select among various arrays that might be available. Within this selected array, the certID in the NV Store area should match.

The actual certificate is built using utility software that could be provided by the platform manufacturer, obtaining some required information in a Host Platform specific way, some from standard TPM commands, and the remaining from this structure in the NV Store area. After building the certificate, the utility should perform a signature verification operation using the appropriate public key to ensure that the certificate was built properly. It is expected, at this point, that the utility would encrypt the entire certificate and store it on the disk. The certificate can, if deleted, be rebuilt at any time in the future.

End of informative comment

Definition

```
typedef struct tdTCG_PARTIAL_SMALL_CERT {
   TCG_PCCLIENT_STRUCTURE_TAG tag;             //   2 bytes
   BYTE                       certType         //   1 byte
   UINT16                     certFlags;       //   2 bytes
   BYTE[4]                    certID;          //   4 bytes
   UINT16                     signatureSize;   //   2 bytes
   BYTE[]                     signature;       //   256 bytes
                                               //   (assuming 2K key)
   UINT16                     additionalDataSize;   //    2 bytes
   BYTE[]                     additionalData;  //      (E.g., serial #)
} TCG_PARTIAL_SMALL_CERT;                      // Minimum total 269 bytes
```

Table 10: TCG_PARTIAL_SMALL_CERT Structure

Type	Name	Description
TPM_STRUCTURE_TAG	tag	MUST be TCG_TAG_PCCLIENT_PART_SMALL_CERT.
BYTE	certType	An element from the TPM_CERT_TYPE table.
UINT16	certFlags	Descriptive information from the TYPM_CERT_FLAGS table.
BYTE[4]	certID	Certificate template ID for 'fixed' information in the certificate, used to select from among a list of cert templates on a CD, disk, or web site.
UINT16	signaturesSize	The size of the signature in bytes
BYTE[ ]	signature	The signature element of the certificate
UINT16	additionalDataSize	The size of the additional data.
BYTE[ ]	additionalData	Additional data necessary to build the cert, usage, and size based on the selected template

8. State Transitions

8.1 Architecture and Definitions

Start of informative comment

A handoff to an operating system generally occurs after the BIOS has completed its initialization and testing of the Host Platform hardware. The BIOS searches through a predefined sequence of boot devices looking for an operating system. A TCG-enabled BIOS will load IPL Code, check for its ability to boot, and just before jumping to the MBR, perform a hash on the code contained within the first 512 bytes where the master boot record is located. This hash will then be extended into PCR[4]. In general, any code that is loaded and jumped to from the BIOS must be hashed and extended into PCR[4] prior to turning control of the system over to that code. The BIOS MUST not hash any data areas.

There can be a number of entries in the boot sequence, but until a bootable device is found, the MBR is not hashed. Just before jumping to the operating system and after the MBR has been loaded into memory, this code must be hashed and extended into PCR[4]. If the operating system returns back to the BIOS through an INT 18h call, the next boot device is checked for boot ability and the process repeats. Each time PCR[4] is extended, an entry is added to the log.

Another item of concern during this phase of a TCG-enabled Host Platform is the passing of the root of trust. The BIOS will pass the chain of trust to the MBR and it is up to the MBR to preserve the chain of trust and pass it to the operating system.

End of informative comment

8.2 Procedure for Transitioning from the Pre-Operating System State and Operating System Present State

Start of informative comment

The platform performs a number of steps in order to make the transition from the Pre-Operating System State to the Operating System Present State,. This section outlines and describes these steps.

End of informative comment

8.2.1 Extending PCR[4] – The IPL Code

Start of informative comment

Just before passing control of the Host Platform to the operating system, the BIOS needs to perform several actions in order to assure that trust in the Host Platform has been maintained. One of the important events that needs to occur is to measure the IPL Code.

End of informative comment

8.2.2 Extending PCR[5] – IPL Configuration and Data

Start of informative comment

PCR[5] is reserved for any configuration data that various transition codes may need. For example, if a BIOS transfers control to an MBR on a hard drive, the MBR may contain optional boot options such as a choice of operating systems or other parameters. This information is measured into this PCR. This PCR may be utilized and extended by any boot loader for variable data.

End of informative comment

8.2.3 Logging of Boot Events

Start of informative comment

A log is provided in memory for the eventual challenger to make a determination of the state of trust of the Host Platform. The operating system handoff code needs to fill this log with information about the boot devices used to get to an operating system. The BIOS functions designed for hashing and extending the various PCRs should automatically log the extending events.

While extending of PCRs is an automatic process in regards to logging, there are other events and information that this specification will require. The following is a list of common log entries that should be recorded prior to turning over control of the system.

1. The type of device that was booted to and specific information about this device, which can uniquely identify it within the system.
2. Each attempt to a boot device should measure:
    a. An EV_SEPARATOR event to all Pre-Operating System State PCRs. (i.e., PCR[0-7]) to delimit the Pre-Operating System State and the Operating System Present State events.
    b. The boot device that is attempted
3. If a boot device returning back to the BIOS through INT 18h or INT 19h affects the transitive trust chain and must therefore be measured.

End of informative comment

1. Upon selecting a boot device, the BIOS measures the EV_ACTION index 0 “Calling INT 19h” into PCR[4].
2. Measure EV_SEPARATOR into PCR[0-7].
3. Measure event type = EV_ACTION, Action index 1 “Returned INT 19h” into PCR[4].[6]
4. Measure the selected IPL Code into PCR[4] using event EV_IPL.
5. Measure the data, including the partition table of the IPL region, if applicable, into PCR[5] using event EV_IPL_PARTITION_DATA.
6. Execute IPL Code.
7. If the IPL Code returns control back to the BIOS using INT 19h, the reason for the return MUST be measured as event type = EV_ACTION, Action index 1 “Returned INT 19h”.

[6] See the note in the definition of this action in Section 10.4.3.

8. If the IPL Code returns control back to the BIOS using INT 18h, the reason for the return MUST be measured as event type = EV_ACTION, Action index 2 “Returned INT 18h”.
9. Upon selecting the next boot device, the BIOS must jump to step 4.

The data within the event field of the EV_SEPARATOR measurement MAY be either a fix or a random value. This may be determined by the Host Platform manufacturer or by a utility under control of the Owner.

8.2.4 Passing Control of the TPM from Pre-Operating System State to the Operating System Present State

Start of informative comment

Once the BIOS has turned control over to an operating system, the Post-boot environment will load its own set of drivers and code to access the TPM. This could cause a potential conflict since there may be contention between the Pre-Operating System State’s and the Operating System Present State’s use and access of the TPM.

The INT 1Ah interface provides for a solution to this problem through the TCG_ShutdownPreBootInterface function. Once the Post-boot environment has loaded its driver support, it MAY call this function to disable the BIOS support. Such a handoff procedure allows for the BIOS support to remain on non-TCG aware operating systems and removes the contention of the TPM hardware on TCG aware operating systems.

If the Operating System is to use the transitive trust chain beyond the measurements indicated in this specification, it is expected that the IPL will continue the measurement of the boot process.

End of informative comment

The IPL Code, regardless of its source, is responsible for measuring:

1. Its parameters into PCR[5].
2. Any code it executes into PCR[4] if it is a “chained” IPL.
3. If is part of an operating system loader, it MAY measure components into PCR[>7].

8.2.5 Various Boot Devices and Special Treatment They May Receive

8.2.5.1 BIOS Aware IPL Devices (BAID)

Start of informative comment

These are devices such as floppy drives, hard drives, CD-ROM drives, etc. The IPL Code of these devices will be measured in PCR[4] just before jumping to this code.

End of informative comment

8.2.5.2 Legacy IPL Devices

Start of informative comment

Option ROMs will have already been measured. INT 18h and INT 19h events will be measured as events per Section 10.4.3. It is the Option ROM's responsibility to measure any additional code loaded.

End of informative comment

8.2.5.3 Boot Connection Vector (BCV)

Start of informative comment

These include devices such as PnP SCSI cards w/ drive and a Non-PnP card w/ expansion header. These cards generally require two BIOS calls, one to return their capabilities and another call to have them hook INT 13h. For the case of the cards that have an associated local mass storage device (SCSI cards with a bootable hard drive), the BIOS must measure the code portion of the first 512 bytes of the mass storage device into PCR[4].

Application Note: For conventional MBR, the first sector of the physical disk would be loaded into memory at 0:7C00h. Offset 0-1B7h within this image would be measured into PCR[4]. Offset 1B8h-1FFh would be measured into PCR[5] using the method below.

End of informative comment

This is treated as the IPL Image.

8.2.5.4 Bootstrap Entry Vector (BEV)

Start of informative comment

A device (generally a network card) that uses a BEV for booting will require that it is called through INT 19h. The Option ROM will need to measure the initial IPL image obtained from the network to PCR[4] prior to jumping to this code.

End of informative comment

1. The entire runtime image of the Option ROM is treated as the IPL Image even if it was measured as an Option ROM into PCR[2].
2. The Option ROM MUST measure the next executable component into PCR[4]. It MUST measure any appropriate data or configuration information (e.g., the source of the executable) into PCR[5].
3. If the BEV returns to the BIOS via a RETF, the BIOS MUST treat this as if the return was via INT 18h.

8.2.5.5 PARTIES Partition

Start of informative comment

The PARTIES Partition is a hidden partition on the hard drive that BIOS can use for additional storage space and as a virtual drive. In the PARTIES Partition, there is a small section called the BEER. Prior to turning control over to the PARTIES Partition, the BIOS must measure the BEER area into PCR[5].

The partition that is booted to in the PARTIES Partition must also have the initial IPL image code measured into PCR[4] prior to turning control over to this code.

End of informative comment

When executing, this is treated as IPL Code including the measurement of it even if the binary image is already measured into PCR[0].

8.2.5.6 El Torito

Start of informative comment

When the BIOS boots to a CD-ROM device that supports the El Torito specification, it first loads the Booting Catalog. If there is more then one boot image on the CD, the user is then prompted to select the boot image. Using this selection and the Booting Catalog, the BIOS loads the first 512 bytes of the CD-ROM that contains the image into memory and turns control over to this code.

Prior to jumping to this code, the BIOS MUST measure the initial IPL image of the boot image code into PCR[4]. The BIOS code will also measure the entire contents of the boot catalog into PCR[5]. The measurement of the boot catalog will be done prior to the measurement of the initial IPL image.

End of informative comment

8.2.5.7 Legacy Reboot

Start of informative comment

Software can store a jump vector in the BDA, set a bit in CMOS, and then Host Platform Reset the system so that the jump vector will be executed as a boot device. Since this code has already been measured, no further measurement is required.

End of informative comment

8.3 Power States, Transitions, and TPM Initialization

Start of informative comment

Suspend is designed to reduce overall power consumption under software control. For instance, for PCs a power management standard called ACPI (*Advanced Control Power Interface Specification*). This standard defines a set of power states that can change the behavior of a device during sleeping states.

End of informative comment

8.3.1 Definitions and Conditions During Power States

Start of informative comment

These are the Sx states as defined in the ACPI specification. The only transitions allowed are those defined in the ACPI definitions.

Host Platforms and Post-BIOS Operating Systems MUST support ACPI.

End of informative comment

There is no required behavior during any power state provided the Host Platform provides resources (e.g., power) to the TPM to perform its required functions during each state. For example, it is obvious that power must be applied during S0. However, for example, the TPM is allowed to be implemented such that auxiliary power is required to maintain PCR registers during S3. In this case, a Host Platform incorporating this type of TPM MUST provide necessary power to the TPM during S3, while Host Platforms incorporating TPMs using flash memory or other NV storage technology will not require power during S3 for this purpose. Another example is the tick counter. There is no requirement for the TPM to maintain this counter across any power state (besides S0, of course). Some TPM manufacturers may choose to provide this feature as a “value add”. Those manufacturer’s TPMs may require auxiliary power during non-S0 power states.

The ACPI “G” states are outside the scope of this specification.

8.3.1.1 S1: Stand-by - Low Wakeup Latency Sleeping State

TPM State:	Fully working, because the TPM is still under power during S1 sleep state.
Entering S1:	Nothing to do.
Exiting S1:	Nothing to do.

8.3.1.2 S2: Stand-by with CPU Context Lost

TPM State:	Fully working, because the TPM is still under power during S2 sleep state.
Entering S2:	Nothing to do.
Exiting S2:	Nothing to do.

8.3.1.3 S3: Suspend to RAM

Start of informative comment

Entering into and exiting from the S3 state is a coordinated effort between the operating system and the BIOS. Since there is no mandated behavior for the TPM during the various power states, this design and protocol assumes the TPM has only two power states: ACPI D0 and ACPI D3. There is no requirement for the TPM to sense any of the normal Host Platform alerts indicating a transition into the S3 power state. Nor is there a requirement for the TPM to sense the normal Host Platform alerts indicating a transition from the S3 power state into the S0 power state. It is therefore a requirement that the Host Platform BIOS and operating system participate in notifying the TPM of these transitions.

The operating system driver notifies the TPM that it is about to transition from the S0 to the S3 power states by sending the TPM_SaveState command. This notifies the TPM that it is to save the required states into non-volatile memory. Upon resume from the S3 power state, the TPM must be notified whether to restore a previously saved state or perform normal initialization. This is done using the TPM_Startup command. The initial state of the TPM can only be determined by the components of the RTM; therefore, the S-CRTM must make the determination as to whether the Host Platform is resuming from the S5/S4 power state or the S3 power state. The S-CRTM must issue the TPM_Startup command with the appropriate parameter indicating to the TPM whether to initialize or restore the previously saved state of the TPM.

It should be noted that if TPM_Startup is issued with the parameter to restore the state of the TPM, when there is no state to restore, the TPM enters a failure mode making it unavailable for the remainder of the power state. The only way to restore the TPM to a functional state is to perform a TPM_Init followed by a TPM_Start indicating to the TPM to clear its internal state.

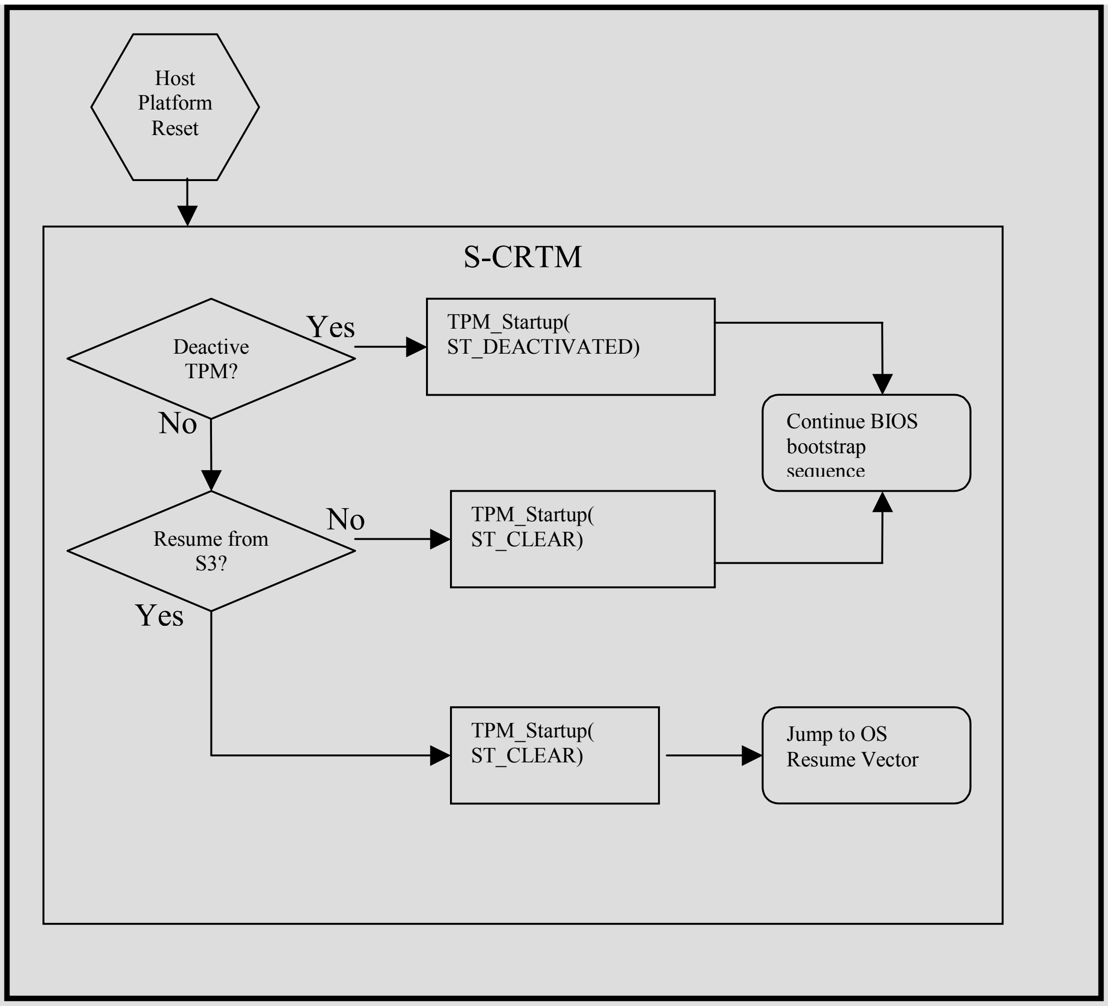


Figure 4: Issuing TPM_Startup

1540 **End of informative comment**

TPM State: The S3 state is the most complex mode to manage, because PCR values and other TPM states MUST be preserved by the TPM during this mode. During S3, the TPM must prohibit all TPM functions.

Entering S3: The operating system driver MUST issue the TPM_SaveState.

1545 **Exiting S3:** The command to restore the PCRs MUST be issued by the CRTM. The BIOS SHOULD ignore any error resulting from the TPM entering failure mode. If any component is executed prior to jumping the operating system resume vector, it MUST have been previously measured before entering the S3 state.

8.3.1.4 S4 Operating System: Suspend To Disk

TPM State: The power to the TPM MAY be removed.

Entering S4: Nothing to do.

Exiting S4: The PCRs will be lost, including the PCRs used by the operating system; therefore, the operating system must establish new integrity. The operating system, therefore, cannot attest to its original power-on state. The S-CRTM MUST issue a TPM_Startup(state = ST_CLEAR).

8.3.1.5 S4 BIOS: Suspend To Disk

TPM State: The power to the TPM MAY be removed.

Entering S4: Nothing to do.

Exiting S4: The PCRs will be lost, including the PCRs used by the operating system; therefore, the operating system must establish new integrity. The PCR contents may be different from S4 from operating system. The S-CRTM MUST issue a TPM_Startup(state = ST_CLEAR).

8.3.1.6 S5: Off State

TPM State: The power to the TPM MAY be removed.

Entering S5: Nothing to do.

Exiting S5: The PCRs will be lost, including the PCRs used by the operating system; therefore, the operating system must establish new integrity. The S-CRTM MUST issue a TPM_Startup(state = ST_CLEAR).

8.3.2 Power State Transitions

Start of informative comment

Each section below describes the behavior and process between the various power states.

End of informative comment

The following pseudo-code represents a set of implementations that generalizes the control flow of the motherboard during the Pre-Operating System State. Not all conditions and error states are included. This is intended only as a guide.

8.3.2.1 S5 → S0

Start of informative comment

This is the transition from a power-off state to a power-on state. Host Platform Reset is asserted. The full BIOS initialization sequence is executed.

End of informative comment

Starting from a power off state.

```
   MAInitTPM (stType = TPM_ST_CLEAR)
   if (MAInitTPM returned OK)
   {
      MAHashAllExtendTPM(CRTM version, PCR[0])
   }
   else  // MAInitTPM returned Error
MAInitError:
   {
      if (PMInitCRTM() indicated TPM failure)
      {
         // Keep communication path open.
         GoTo POST_BIOS     // Transfer control to POST BIOS.
      }
      else // Assume communication path failed
      {
         if (Disable TPM Interface is provided)
         {
            Disable Interface to TPM
         }
         else
         {
            Disable the Host Platform
         }
      }
}
   if (Normal boot)
   {
      MAHashAllExtendTPM(Initial POST BIOS, PCR[0])
      GoTo POST_BIOS     // Transfer control to POST_BIOS
}
```

```
// Note: the following else cluase is optional depending if either the
// BIOS Recovery Mode or a Utility requiring physical presence
// indication from the boot state is part of the motherboard’s design.
else if (executing BIOS Recovery Mode)
{
      MAHashAllExtendTPM(BIOS Recovery Code, PCR[0])
      GoTo BIOS_Recovery_Code
}
else if (indication of physical presence given to BIOS)
{
      if (Host Platform requires physical presence during
            boot state)
      {
         MAHashAllExtendTPM(Utility, PCR[0])
         MAPhysicalPresenceTPM(   TPM_PC_PHYSICAL_PRESENCE_MASK_SW |
                                  TPM_PC_PHYSICAL_PRESENCE_PRESENT)
         GoTo Physical_Presence_Utility
      }
}

POST_BIOS:
   TCG_StatusCheck()
   Optionally TCG_PassThroughToTPM(TPM_DisableOwnerClear)
   Optionally TCG_PassThroughToTPM(TPM_DisableForceClear)

   If (Embedded Option ROMs)
      TCG_HashLogExtendEvent(Embedded Option ROMs, PCR[0])

   TCG_HashLogExtendEvent(Host Platform Configuration, PCR[1])

   While (Unexecuted Option ROM present)
   {
      TCG_HashLogExtendEvent(Visible Portion of Option ROM, PCR[2])
      Transfer control to Option ROM.
   }

INT_18:
   Choose next IPL Image
TCG_HashLogExtendEvent(“Calling INT 19h”, PCR[4])
   INT 19h // To Execute IPL Code

INT_19:
   TCG_HashLogExtendEvent(Chosen IPL Code, PCR[4])
   TCG_HashLogExtendEvent(IPL data and partition info, PCR[5])
TCG_HashLogExtendEvent(EV_Separator, PCR[0-7])
Jump to IPL Code

IPL Code Entry point:
TCG_HashLogExtendEvent(IPL Configuration Data, PCR[5])
Transfer Control to the operating system Loader
   if (the operating system loader fails to load the operating system)
GoTo INT_18
```

```
BIOS_Recovery_Code:
Transfer control of Host Platform to BIOS Recovery Code
   When complete perform Host Platform Reset

Physical_Presence_Utility:
Transfer control of Host Platform to Utility Requiring Physical Presence
When complete perform Host Platform Reset

END
```

8.3.2.2 S1 → S0

Start of informative comment

Resume from an S1 suspend state. Host Platform Reset has never been asserted so the TPMInitCRTM function cannot be called. The S-CRTM executes code to perform the resume operation without re-measuring the Pre-Operating System State. The CRTM then passes control directly to the Operating System Present State. If there are any changes to the Host Platform's components or configuration, measuring these changes is the responsibility of the Operating System Present State components.

End of informative comment

No Action

8.3.2.3 S2 → S0

Start of informative comment

Resume from an S2 suspend state. Host Platform Reset has never been asserted so the TPMInitCRTM function cannot be called. The CRTM executes code to perform resume without re-measuring Pre-Operating System State. CRTM passes control directly to the operating system. If there are any changes to the Host Platform's components or configuration, measuring these changes is the responsibility of the Operating System Present State.

End of informative comment

No Action

8.3.2.4 S3 → S0

Start of informative comment

Resume from an S3 suspend state. Host Platform Reset is asserted. The CRTM executes code to perform resume without re-measuring Pre-Operating System State. The CRTM passes control directly to the operating system. If there are any changes to the Host Platform's components or configuration, measuring these changes is the responsibility of the operating system.

The operating system must assure prior to entering S3 that the TPM has preserved the required values.

There must be a countermeasure in the event POST is modified by malicious code and the Host Platform resumes from S3 executing that code. After modifying the BIOS, the operating system is required to transition the Host Platform to S5 before allowing a transition to S3.

This is to allow the new BIOS to be measured. The CRTM is responsible for enforcing this behavior.

End of informative comment

If the S-CRTM executes code outside the S-CRTM prior to jumping to the operating system S3 resume vector, CRTM MUST be able to determine if there has been an update to any portion of the BIOS since the previous transition from S5. If the CRTM detects a modification to BIOS since the last transition from S5, the CRTM MUST either:

1. Force the Host Platform to transition to S5
2. Make the contents of PCR[0] invalid

```
MAInitTPM (stType = TPM_ST_STATE)
If MAInitTPM returned OK
{
   If BIOS modified since last S5
   {
      Force transition to S5.
      or
      Invalidate PCR[0].
   }
   Transfer control to the operating system.
}
else
{
   Force transition to S5.
   GoTo MAInitError in Section 8.3.2.1
}
```

8.3.2.5 S4 → S0

Start of informative comment

This is where the IPL Code, instead of loading the operating system loader, will load memory from a hard disk. Host Platform Reset is asserted. The full BIOS initialization sequence is executed just like the S5->S0 transition. If there are any changes to the Host Platform's components or configuration, measuring these changes is the responsibility of both the Pre-Operatig System State and the Operating System Present State.

End of informative comment

Same as S5->S0 except IPL Code loads the saved memory image.

9. ACPI Device Object for the TPM

Start of informative comment

In order to facilitate device discovery and operating system driver loading, the platform's ACPI name space contains an appropriate device object for the TPM. This device scope appears in the appropriate bus hierarchy; i.e., within the scope of the LPC bus. The TPM device also contains, at a minimum, a _HID object and resource descriptors to claim all hardware resources consumed by the TPM. The example below shows a minimal snippet of typical ASL.

```
Device (TPM) {
   Name (_HID, EISAID("PNP0C31"))
   Name (_CRS, ResourceTemplate() {
         Memory32Fixed (ReadWrite, 0xFED40000, 0x5000,)
   })
}
```

If legacy I/O ports or any other hardware resources are decoded by the TPM, they are declared here also. Additionally, a _CID may be included. Per the ACPI specification, a _CID may be a single ID or a package of IDs listed in order of preference.

The example device object below shows a vendor specific _HID to facilitate loading of a vendor specific driver. The generic TPM 1.2 PNP ID is given in the _CID object. There are also legacy I/O ports declared in this example device object.

```
Device (TPM) {
      Name (_HID, EISAID("IFX0101"))
      Name (_CID, EISAID("PNP0C31"))
   Name (_CRS, ResourceTemplate() {
      Memory32Fixed (ReadWrite, 0xFED40000, 0x5000,)
      IO (Decode16, 0x004E, 0x004E, 0x01, 0x02) //IO config 4Eh-4Fh
      IO (Decode16, 0x4700, 0x4700, 0x01, 0x0C) //IO runtime 4700h-470Ch
   })
}
```

End of informative comment

The platform ACPI namespace MUST contain an ACPI device object in an appropriate scope for the TPM. This device object MUST contain either a _HID object with the value of "PNP0C31", a _CID object with the value of "PNP0C31", or a _CID object that evaluates to a package where the value "PNP0C31" is one of the IDs within the package.

The ACPI device object representing the TPM MUST claim all hardware resources consumed by the TPM. This includes any legacy I/O ports and other hardware resources.

If there are configurable resource options, the ACPI device object representing the TPM MUST also contain _PRS and _SRS control methods as required by the ACPI specification.

10. Event Logging

10.1 Byte-ordering (Endian-ness)

Start of informative comment

TPM data and structures, are big-endian. However, the processor in the PC Client represents data in little-endian format so all constants and data created and used by the PC Client's structures shall be in little-endian format.

The justification for this is: when software deals with Host Platform itself (e.g., the PC Client data, structures, etc.), it does little-endian - always. Software already deals with this bifurcation when communicating over a network. When getting packets from the network (e.g., FTP, HTTP, etc.), it deals with big-endian; when it deals with data and structures from the Host Platform itself, it does so using little-endian format. What would be inconsistent is if we changed within the context or purview.

End of informative comment

1. All constants and data SHALL be represented as little-endian unless otherwise explicitly stated.
2. All strings SHALL be represented as an array of ASCII bytes with the left-most character placed in the lowest memory location.

10.2 ACPI Table Usage

Start of informative comment

A system's firmware uses an ACPI table to identify the system's TCG capabilities to the Post-boot environment. The information in this table is not guaranteed to be valid until the BIOS performs the transition from the Pre-Operating System State to the Operating SystemOperating System Present State.

The firmware "pins" the memory associated with the Pre-Operating System State's TCG log and reports this memory as "Reserved" memory via the IA32 INT 15h/E820 interface. This is done to ensure that the log area contains the EXTEND operations performed on the most recent system transition from S5 or S4. If the log were in reclaimable memory, the firmware would not be able to report the system configuration on the return from hibernation (S4) since the memory would have been reclaimed for other use by the operating system on its last boot from S5.

Note: The character string "TCPA" is used for compatibility with previously defined structures.

End of informative comment

The ACPI table indicated in Figure 5 as “TCPA” is defined in the *TCG ACPI Specification; Specification Version 0.0; Revision 0.9; February 14, 2005.*

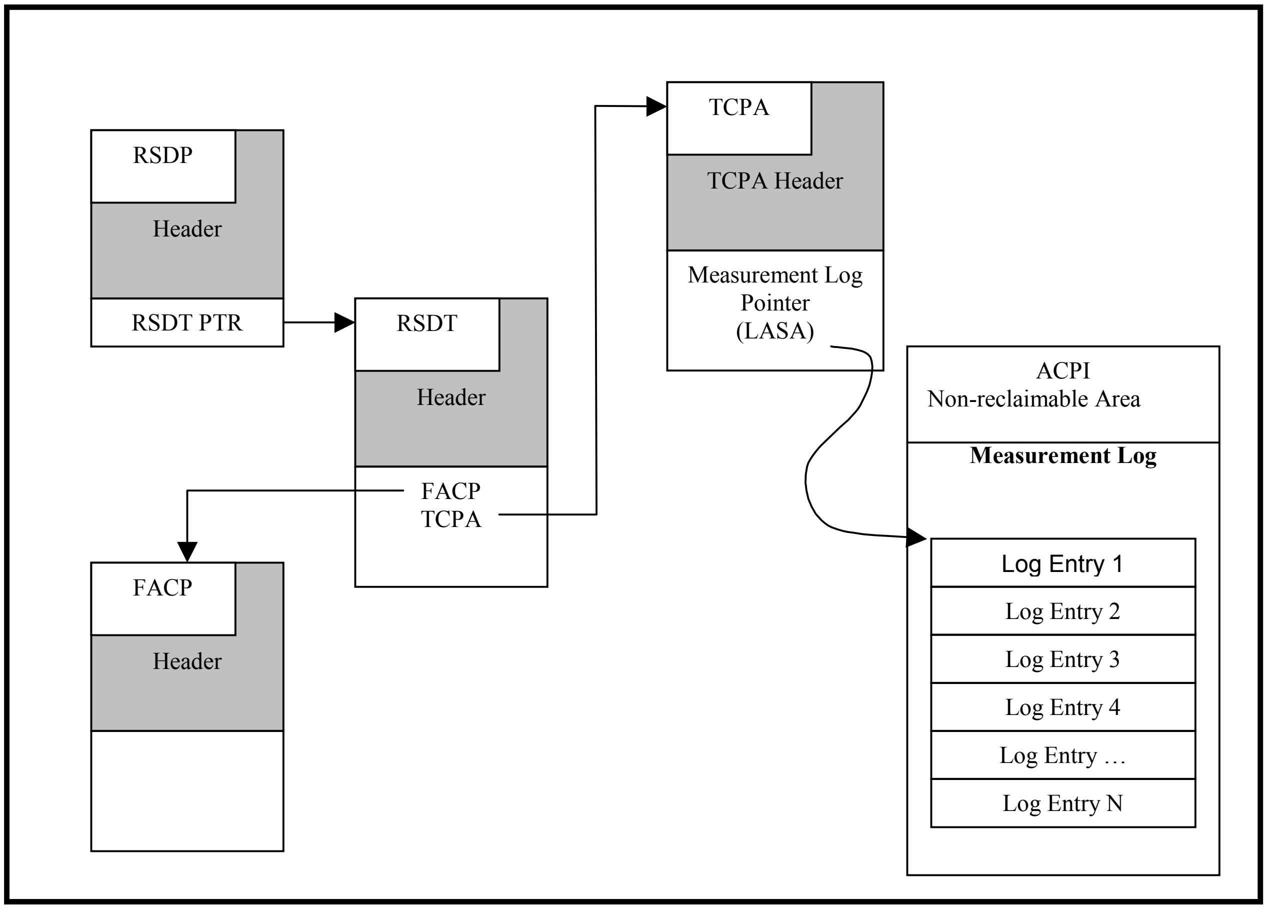


Figure 5: ACPI Table

10.2.1 TCG_PCClientPCREventStruct Structure

Start of informative comment

In Version 1.1 of this specification, this structure was defined by the *TPM Main Specification*. In the Version 1.1 view of the TCG architecture, these structures were to be taken “as is” by the Host Platform’s software (e.g., the TSS), stored, and transferred unmodified by the infrastructure to the challengers. It is unlikely that an infrastructure and challengers will want, or even be able to accept, the same format from all classes of Host Platforms. A better method is to require the Host Platform’s operating system-present software to support a Host Platform-specific structure and have that software perform whatever conversions are necessary to conform to the requirements of the infrastructure and challengers. For example, some challengers may accept raw ‘C’-type data structures, while others require XML formatted data. Therefore, the notion that this structure is to be defined by specifications outside this one is no longer valid. This event structure is therefore considered to be PC Client platform-specific. It is the responsibility of the Host Platform’s software (e.g., a TSS) to understand this structure and convert it to any relevant format. Because this structure contains cryptographic information (i.e., hash values), it is important to understand what can and cannot be converted or altered.

Each "Measurement Log" entry in the ACPI table shown in Figure 5 is an instance of this TCG_PCClientPCREventStruc structure.

End of informative comment

Each entry in the Event Log SHALL use the following structure:

```
TCG_PCClientPCREventStruc
   pcrIndex      DD ?
   eventType     DD ?
   digest        DB 20 dup (?)
   eventDataSize DD ?
   event         DD ?
TCG_PCClientPCREventStruc
```

```
pcrIndex      The PCR Index to which this event was extened.

eventType     Defined in Section 10.4.1.

digest        The value extended into the PCR identified by pcrIndex.

eventDataSize The size of the event.

event         The data of the event.
```

10.3 Measurement Event Entries and Log

Start of informative comment

The value within a PCR is used both for sealed storage and for Attestation. When used for Attestation, the raw hash value carries little meaning. Therefore, more meaningful structures are used that carry with them information. This information is contained within the structure TCG_PCClientPCREventStruc described in Section 10.2.1.

The procedure for creating the measurement is to take an SHA-1 hash of the data contained within the TCG_PCClientPCREventStruc.event field which in practice is the entire structure PlatformSpecificEventLogStruct. The resulting hash is placed into the field TCG_PCClientPCREventStruc.digest and used as the data in the TPM_Exend operation.

These structures are stored as an unstructured array within the ACPI data area as defined in Section 10.2. None of the Pre-Operating System State's entities, including ACPI, are required to interpret this data. The storage of this data using ACPI is a convenience because there are defined mechanisms already in place to allow the transfer of this information to the Post-boot State. Once the Post-boot State controls the Host Platform, the Post-boot Operating System is expected to read this data and transfer it to its own event log.

Due to the characteristics of the hashing operation, the verification of the Measurement Log entries is order dependent. This, by the way, is a beneficial characteristic by adding trust in the order of the events. That is, if measurement A is taken and a Measurement Log for Measurement Log Entry A is created; followed by a measure B and a Measurement Log entry B is created; it is important to a verifier that the sequence of Measurement Log Entry A and Measurement Log Entry B be deterministic.

End of informative comment

1. The hash used MUST be SHA-1 as specified in the *TPM Main Specification, Version 1.2*.
2. Procedure:
   a. Set A to an instantiation of a TCG_PCClientPCREventStruc structure.
   b. Set A.pcrIndex to the index of the PCR to be extended.
   c. Set A.eventType to the specified eventType defined in Section 10.4.1.
   d. Fill A.event with either the data to be measured or the description of data to be measured.
   e. A.digest = SHA-1{A.event}.
   f. Extend A.pcrIndex using A.digest as the value.

   Note for Step d: The description used here is not to be taken literally for all event types. Where the event field will contain data or structures, this statement is to be taken literally and the event field is to be filled in. However, when measuring code, it is obviously not practical to place the entire code area into the event field just to take the hash of it. Also, it is not expected for the event field to contain the entire code area in the event log for these event types. For precise contents of this field, refer to the description of the event type.
3. For each PCR, the sequence of the Measurement Log Entries MUST be in the same sequence that the events we extended into the TPM.

10.4 Event Descriptions

Start of informative comment

There are two classes of event types: Host Platform Independent and Host Platform Specific. Host Platform Independent event types contain information, which is generally applicable to all classes of Host Platforms so they are defined in the *TCG Main Specification*. Most event types, however, apply only to a particular class of Host Platform so they are designated as Host Platform Specific and the each Platform Specific Specification (e.g., this document) defines them using the event type EV_PLATFORM_SPECIFIC to indicate the platform class.

End of informative comment

10.4.1 Event Types

Table 11 lists the events that are defined for the field TCG_PCClientPCREventStruc.eventType. Previous versions of this specification depended on other TCG specifications to define the event types. This specification takes over the role of defining the event types but for backward compatibility preserves the meaning of the original values, only changing the label of the event to provide a better description. Also, some event tags defined in previous versions of this specification have now been converted to event types. Those also, are listed in the third column of Table 11.

Table 11: Event Types

Label	Value	Previous Label or Event Tag[7]	Description
EV_PREBOOT_CERT	00h	EV_CODE_CERT	The event field contains certificates such as the Validation Certificates.

EV_POST_CODE	01h	EV_CODE_NOCERT
		POST Contents;
		EventID = 000Dh
			The digest field contains the SHA-1 hash of the POST portion of the BIOS. The event field SHOULD NOT contain the actual POST code but MAY contain informative information about the POST code.

EV_UNUSED	02h	EV_XML_CONFIG	The event type was never used and is considered reserved.
EV_NO_ACTION	03h	EV_NO_ACTION	The event field contains informative data that was not extended into any PCR. The fields: pcrIndex and digest MUST contain the value 0.
EV_SEPARATOR	04h	Same	Delimits actions taken during the Pre-Operating System State and the Operating System Present State.
EV_ACTION	05h	Same	A specific action measured as a string defined in Section 10.4.3.
EV_EVENT_TAG	06h	EV_PLATFORM_SPECIFIC	The event field contains the structure defined in Section 10.4.2.1.

EV_S_CRTM_CONTENTS	07h	S-CRTM Contents;
		EventID = 000Ch
			The digest field contains is the SHA-1 hash of the S-CRTM. The event field SHOULD NOT contain the actual S-CRTM code but MAY contain informative information about the S-CRTM code.

EV_S_CRTM_VERSION	08h	S-CRTM Version String;
		EventID = 000Bh
			The event field contains the version string of the S-CRTM.

EV_CPU_MICROCODE	09h		The event field contains a descriptor of the microcode but the digest field contains the actual hash of the microcode patch that was applied.
EV_PLATFORM_CONFIG_FLAGS	0Ah	None	The format and contents to be defined by the platform manufacturer. Examples of information contained in this event type are the capabilities of the platform's measurements, whether the Owner has disabled measurements, etc.

EV_TABLE_OF_DEVICES	0Bh	Platform Manufacturer Table of Devices;
		EventID = 000Eh
			The event field contains the Platform manufacturer-provided Table of Devices or other Platform manufacturer-defined information. The Platform manufacturer defines the content and format of the Table of Devices. The Host Platform Certificate may provide a reference to the meaning of these structures and data. This structure is measured into PCR[1] using the following.

EV_COMPACT_HASH	0Ch	None	This event is entered using the TCG_CompactHashLogExtendEvent. While it can be used by any function, it is typically used by IPL Code to measure events. The contents of the event field is specified by the caller but is not part of the measurement; rather, it is just informative.

[7] Note: The contents of this column are not intended to be part of the event. They are only for the reader's convenience when referencing the equivalent event types or eventIDs of the *PC Client Specification, Version 1.1*.

Label	Value	Previous Label or Event Tag[7]	Description
EV_IPL	0Dh	None	The digest field contains the SHA-1 hash of the IPL Code. The event field SHOULD NOT contain the actual IPL Code but MAY contain informative information about the IPL Code. Note: The digest may not cover the entire area hosting the IPL Image, but only the portion that contains the IPL Code. For example, if the IPL Image is a disk drive MBR, this MUST NOT include the portion of the MBR that contains the disk geometry.
EV_IPL_PARTITION_DATA	0Eh	None	The data and partition portion of the IPL Image.
EV_NONHOST_CODE	0Fh	None	The executable component of any Non-host Platform. The contents of the event field are defined by the manufacturer of the Non-host Platform.
EV-NONHOST_CONFIG	10h	None	The parameters associated with a Non-host Platform. The contents of the event field are defined by the manufacturer of the Non-host Platform.
EV_NONHOST_INFO	11h		The event is information about the presence of a Non-host Platform. This information could be, but is not required to be, information such as the Non-host Platform manufacturer, model, type, version, etc. The information and formatting is to be determined by the BIOS.

10.4.2 Host Platform Specific Log Events

1. For the events described in this subsection, the field TCG_PCClientPCREventStruc.eventType SHALL be EV_EVENT_TAG.
2. The field TCG_PCClientPCREventStruc.event SHALL be a TCG_PCClientTaggedEventStruct structure.

10.4.2.1 Tagged Event Log Structure

The events shall be the following structure.

```
TCG_PCClientTaggedEventStruct    STRUCT
   EventID       DD ?  / Tag as defined in this subsection below.
   EventDataSize DD ?  / Size of EventData
   EventData     DB ?  / EventData
TCG_PCClientTaggedEventStruct    ENDS
```

10.4.2.2 Special Purpose Structures

10.4.2.2.1 OptionROMExecuteStructure

```
OptionROMExecuteStructure             STRUCT
   Reserved        DW    0         / Reserved
   PFA             DW    ?         / Adapter PFA
   HashData        DB    20dup(?)  / Hash of visible Option ROM code
OptionROMExecuteStructure             ENDS
```

10.4.2.2.2 OptionROMConfigStructure

```
OptionROMConfigStructure              STRUCT
   Reserved         DW    0        / Reserved
   PFA              DW    ?        / Adapter PFA
   OptionROMStruct  DB    ?        / Defined by Device Vendor
OptionROMConfigStruct                 ENDS
```

10.4.2.3 Host Platform Specific Event Tags

As stated in Section 10.1, data is represented in little-endian format.

The references to EventID are TCG_PCClientTaggedEventStruct.EventID.

The references to EventData are TCG_PCClientTaggedEventStruct.EventData.

10.4.2.3.1 SMBIOS Structure

Each event MAY consist of one or more complete SMBIOS records. This event may appear multiple times in the event log. The SMBIOS structure SHALL be logged using the following:

EventID = 0001h

EventData[] = One or more raw complete SMBIOS records.

10.4.2.3.2 BIS Certificate

The BIS Certificate SHALL be logged using the following:

EventID = 0002h

EventData[] = Raw BIS Certificate

10.4.2.3.3 POST BIOS ROM Strings

The BIOS ROM Strings SHALL be logged using the following:

EventID = 0003h

EventData[] = Hash of POST BIOS ROM Strings

10.4.2.3.4 ESCD

The ESCD SHALL be logged using the following:

EventID = 0004h

EventData[] = Hash of ESCD Data

10.4.2.3.5 CMOS

The CMOS SHALL be logged using the following:

EventID = 0005h

EventData[] = Raw CMOS Data

10.4.2.3.6 NVRAM

The NVRAM SHALL be logged using the following:

EventID = 0006h

EventData[] = Raw NVRAM contents

10.4.2.3.7 Option ROM Execute

The BIOS logs the execution of each Option ROM into PCR[2] using the following:

EventID = 0007h

EventData[] = OptionROMExecuteStructure

10.4.2.3.8 Option ROM Configuration

Option ROMs log events into PCR[3] using the following:

EventID = 0008h

EventData[] = OptionROMConfigStructure

10.4.2.3.9 Option ROM Microcode Update

Option ROMs log events into PCR[2] using the following:

EventID = 000Ah

EventData[] = Hash of Microcode that will be loaded.

10.4.2.3.10 S-CRTM Version String

Deprecated by Event Type: EV_S_CRTM_VERSION

CRTM version string log events into PCR[0] using the following:

EventID = 000Bh

EventData[] = S-CRTM version string. This is an opaque value.

10.4.2.3.11 S-CRTM Contents

Deprecated by Event Type: EV_S_CRTM_CONTENTS

CRTM contents log events into PCR[0] using the following:

EventID = 000Ch

EventData[] = Hash of entire S-CRTM.

10.4.2.3.12 POST Contents

Deprecated by Event Type: EV_S_CRTM_POST_CODE

CRTM contents log events into PCR[0] using the following:

EventID = 000Dh

EventData[] = Hash of entire POST code and data

10.4.2.3.13 Host Platform Manufacturer Table of Devices

Deprecated by event type: EV_TABLE_OF_DEVICES

EventID = 000Eh

EventData[] = Table of Device. Structures and data to be defined by the Host Platform manufacturer.

10.4.3 EV_ACTION Event Types

The following action strings are defined. The strings below are enclosed in quotes for clarity; the actual log entries SHALL NOT include the quote characters. They SHALL be logged using the following:

TCG_PCClientPCREventStruc.EventType = EV_ACTION

TCG_PCClientPCREventStruc.Event[] = ASCII string as shown in Table 12

Table 12: EV_ACTION Strings

Action Index[8]	String	Purpose and Comments	PCR
0	"Calling INT 19h"	The BIOS is calling INT 19h. f no additional strings are posted in the log, that means that the software that '"hooked" the INT 19h vector did not return control to the BIOS.	4
1	"Returned INT 19h"	Entering the INT 19h handler. This means either the BIOS has transferred control to the INT 19h handler, or the BIOS received control back from a failed IPL. If the called code is not TCG-aware, it may have loaded additional unmeasured code. However, there is a log entry showing entry to (and measurement of) untrusted code. NOTE: the term "returned" is an anachronism originating from a previously misdefined usage of this event; however, there is no reason to change the string.	4
2	"Return via INT 18h"	The BIOS received control back via INT 18h. If the called code is not TCG-aware, it may have loaded additional unmeasured code. However, there is a log entry showing entry to (and measurement of) untrusted code.	4
3	"Booting BCV Device s"	The BIOS is IPL/Booting a BCV Device. The value "s"is a ASCII string that unambiguously describes the boot device. This SHOULD include an indication of logical or physical device location and any ID string returned by the device.	4
4	"Booting BEV Device s"	The BIOS is IPL/Booting a BEV Device. The value "s" is an ASCII string supplied by the BEV device.	4
5	"Entering ROM Based Setup"	The BIOS is entering ROM-based Setup during Pre-Operating System State.	0

[8] This is only used for reference within this document.

Action Index[8]	String	Purpose and Comments	PCR
6	"Booting to Parties N"	The BIOS is IPL/Booting from a Parties Partition #N. The value "N" is the actual numeric value of the partition number represented as a printable ASCII hex value (e.g., partition zero would get the string value "0"). Where "N" is the index into the BEER table.	4
7	"User Password Entered"	The User has entered the correct user password.	4
8	"Administrator Password Entered"	The User has entered the correct administrator password.	4
9	"Password Failure"	The typed password did not match the stored password after a specified number of retries.	4
10	"Wake Event n"	The cause of the power to be applied to the Host Platform where "n" is the wake source (e.g., wake source zero would get the string value "0"). The value "n" is as specified in the *SMBIOS Specification*, Section 3.3.2.1. Currently only wake from S4 and S5 MUST be supported.	6
11	"Boot Sequence User Intervention"	The User altered the boot sequence.	4
12	"Chassis Intrusion"	The case was opened.	1
13	"Non Fatal Error"	A non-fatal POST error (e.g., keyboard failure) was encountered. This is any error that allows the system to continue the boot process	1
14	"Start Option ROM Scan"	The BIOS has started the Option ROM scan.	2
15	"Unhiding Option ROM Code"	Unhiding Option ROM Code. This is typically done by the visible portion of the Option ROM but is measured into PCR 2 because this PCR measures event related to the code portion of the Option ROMs.	2
16	"<OpRom Specific non-IPL String>"	An Option ROM vendor specific string for non-Boot/IPL events.	3
17	"<OpRom Specific IPL String>"	An Option ROM vendor specific string for Boot/IPL events.	5

11. Implementation Overview

Start of informative comment

2010 The TCG interfaces in the Pre-Operating System State are as follows:

- TCG Application Interface
- TCG Pre-Operating System State Driver Interface
- A set of ACPI structure definitions containing the Event Log

A diagram of these interfaces and their relationship is shown in Figure 6.

2015 **End of informative comment**

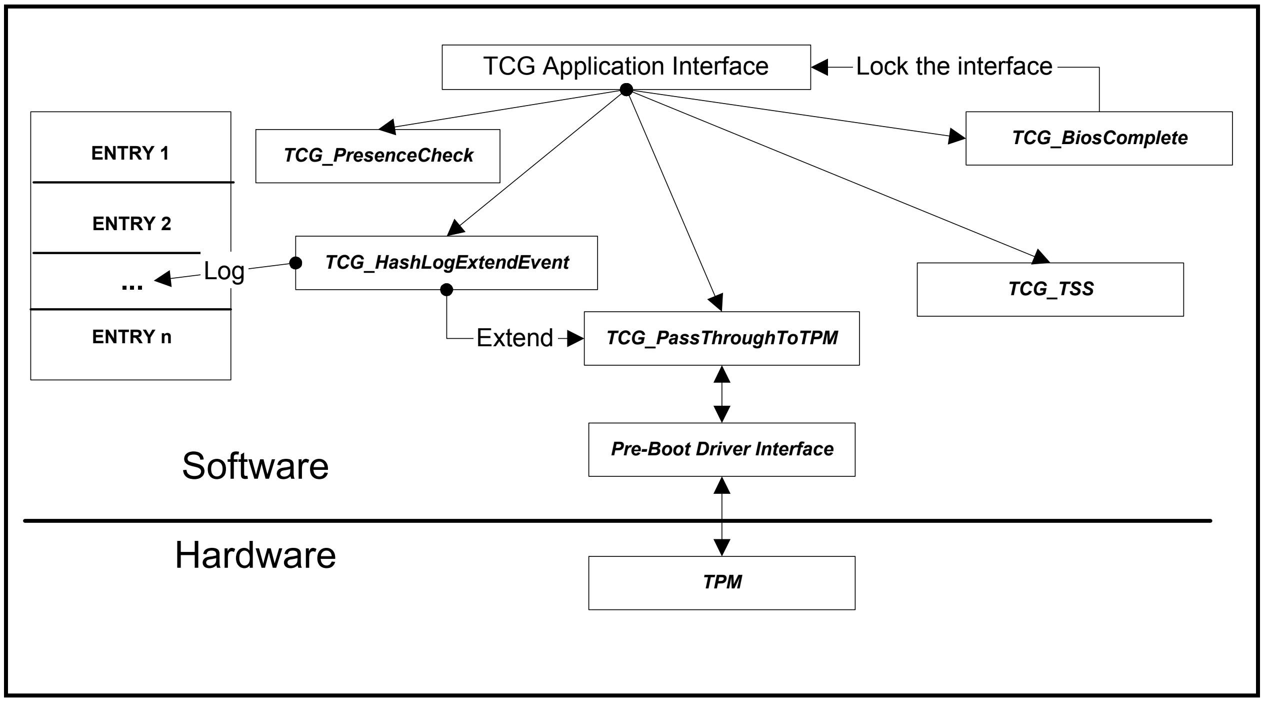


Figure 6: Pre-Operating System State Interfaces

12. Application Level Interface

Start of informative comment

This interface is intentionally lightweight, since the anticipated users of this interface are in the space-restricted Pre-Operating System State, e.g., Option ROM and boot-record code. As a result, an INT 1Ah interface is defined, which allows the caller of the interface to have direct access to a limited set of TSS functions and a pass-through to the TPM.

The Pre-Operating System State's entities may intercept the INT 1Ah interface for the purpose of adding TSS functions. No mechanism is required, nor suggested, for preventing an attack by intercepting the INT 1Ah.

End of informative comment

Upon entry:

1. There MUST be no requirement placed on the A20 state on entry to these INT 1Ah functions.
2. The processor's memory mode into these INT 1Ah functions MUST be in Real Mode and MUST NOT be in virtual 8086 mode.

On exit:

1. These INT 1Ah functions MUST preserve the A20 state and return with the A20 mask unchanged from call.
2. The processor's memory mode MUST be Real Mode with 64K segment limits.

This interface only supports up to 4 GB of physical address space.

Note: The value 41504354h is the character string "TCPA" and is used for compatibility with previously defined structures and interfaces.

12.1 General Calling Convention

Each function below will have the following general calling convention:

On entry:

(AH)	=	BBh
(AL)	=	Function selector, see below
(ES)	=	Segment portion of the pointer to the input parameter block
(DI)	=	Offset portion of the pointer to the input parameter block
(DS)	=	Segment portion of the pointer to the output parameter block
(SI)	=	Offset portion of the pointer to the output parameter block
(EBX)	=	41504354h
(ECX)	=	0
(EDX)	=	0

On return:

(EAX)	=	Return code. If (AH) = 86h, the function is not supported by the system.
(DS:SI)	=	Modified based on specific function called.

All other register contents including upper words of 32-bit registers are preserved. Note that this cannot be guaranteed if (AH) = 86h because the call could be made on a pre-TCG BIOS.

12.2 Indicators of Supported Functions

Start of informative comment

There are two indicators that a function is supported.

The first method is the return within EAX of the value 86h.

The second method is use of the Carry Flag. The Carry Flag is used as an indicator to the caller that the specific function within the INT 1Ah interface is implemented. If the carry flag is returned clear (value = 0), the function is implemented and the return values are valid. If the carry flag is returned set (i.e., value = 1), the function is not implemented and the other return values are not valid.

The function indicators in the range of BBXXh were chosen by researching various informal databases and querying "experts" in the field. Because there is no "registrar" of INT 1Ah functions there is a risk, though small, that a platform's BIOS implements the INT 1Ah functions within this range that do not perform these TCG functions. For this reason, the return of the value 41504354h within EBX from the TCG_StatusCheck provides more assurance to the caller that this range of functions implements this set of TCG functions.

End of informative comment

1. All TCG INT 1Ah function MUST return with the Carry Flag (CF) clear (value = 0).
2. Any TCG INT 1Ah function MUST return with the Carry Flag (CF) set (value – 1).

12.3 Return Codes

The defined error codes the Pre-Operating System State functions MAY return are listed in Table 13.

Table 13: Return Codes

Return Code	Value	Description
TCG_PC_OK	0000h	The function returned successful.
TCG_PC_TPMERROR	TCG_PC_OK + 01h | (TPM driver error << 16)	The TPM driver returned an error. The upper 16 bits contain the actual error code returned by the driver as defined in Section 13.
TCG_PC_LOGOVERFLOW	TCG_PC_OK + 02h	There is insufficient memory to create the log entry.
TCG_PC_UNSUPPORTED	TCG_PC_OK + 03h	The requested function is not supported.

12.4 Parameter Block

Input and output data is formatted in parameter blocks. The first entry (labeled either IPBLength or OPBLength) in the parameter block is a WORD-sized value that contains the size of the parameter block inclusive of the size entry. The second entry (labeled Reserved) is a reserved WORD that MUST contain the value 0h. Entries, if any, after the reserved entry are defined by the particular interface.

The caller of the INT 1Ah interface must allocate OP Buffer of at least 4 bytes long. (Can be 2 bytes since OPBLength field is defined as WORD, but I am assuming that the following “Reserved” WORD was abutted just to keep alignment); otherwise, the INT 1Ah interface behavior is undefined and may cause system crush.

On an error, the INT 1Ah interface MUST return an output parameter block. In most error cases, this will contain an OPBLength of 0004h to indicate the allocation of the OPBLength and the Reserved entries.

For the TCG_PassThroughToTPM function, it is not an error case if the OPBLength value supplied by the caller in the input parameter block does not specify an output buffer size large enough to hold the entire command result returned to the BIOS from the TPM. In this case, the BIOS MUST truncate the result returned from the TPM when the output buffer allocated by the caller is full and then set the OPBLength value in the output parameter block to reflect that.

NOTE: For all the functions defined for this interface, the BIOS MUST set the OPBLength value in the output parameter block to the length of the data it returns to the caller in the output buffer, including the 4 byte entries that must be at the beginning of the output buffer.

12.5 TCG_StatusCheck

INT 1Ah (AH)=BBh, (AL)=00h

This function call verifies the presence of the TCG BIOS interface and provides the caller with the version of this specification to which the implementation complies. The TCG BIOS interface code MAY call the MP driver function MPInitTPM during the first invocation of the TCG_StatusCheck function.

To allow a caller to know the status of the TPM and the status of this interface, the following return codes are defined. If either of these cases occurs, the function is to be considered to have succeeded and the values in the registers below MUST contain proper values.

1. If the TPM is not present, this function MUST return the TCG_PC_TPM_NOT_PRESENT error.
2. If the TPM is present but deactivated by TPM_Init or TPM_Startup, this function MUST return the TCG_PC_TPM_DEACTIVATED error. Any entity may temporality deactivate the TPM subsequent to TPM_Init (e.g., calling TPM_TempDeactivate using the pass through function). In this case, this function will not return this error even though the TPM is deactivated because this return code is an indicator of the TPM’s state during TPM_Init. This return code can, therefore, be used as an indicator as to whether there are entries in the event log.

On entry:

(AH)	=	BBh
(AL)	=	00h

On return:

(EAX)	=	Return code. Set to 00000000h if the system supports the TCG BIOS calls.
(EBX)	=	41504354h
(CH)	=	TCG BIOS Major Version (01h for Version 1.0)
(CL)	=	TCG BIOS Minor Version (02h for Version 1.2)
(EDX)	=	BIOS TCG Feature Flags (None currently defined. MUST be set to 0)
(ESI)	=	Absolute pointer to the beginning of the Event Log
(EDI)	=	There are three conditions that need to be accommodated: 1. If the Event Log is empty (i.e., no PCClientPCREventStruc structures), EDI MUST be set to 0. 2. If there is exactly one PCClientPCREventStruc structure, EDI MUST be set to ESI. 3. If there is more than one PCClientPCREventStruc structure, EDI MUST be set to the absolute pointer to the first byte of the last event in the log.

Note: The caller must assume that no registers are preserved by the call, since the call might be made in an unsupported system environment.

12.6 TCG_HashLogExtendEvent

INT 1Ah, (AH)=BBh, (AL)=01h

This function performs hashing of the event or the event data, extends the event to a PCR, then places the resulting TCG_PCClientPCREventStruc into the event log. The caller should verify the availability of this function by issuing a previous call to the TCG_StatusCheck function, that way the caller can be assured that calls to this function preserve the register contents (including the upper 16 bits of 32-bit registers).

If this function cannot create a Measurement Log Entry (e.g., the Measurement Log is full), this function MUST perform the TPM_Extend operation.

Description of Process

If either input parameter HashDataPtr or input parameter HashDataLen is not NULL, this function MUST perform the following:

1. Treat the input parameter LogDataPtr as a completed TCG_PCClientPCREventStruc except the TCG_PCClientPCREventStruc.digest field.
2. Perform the hash function on the TCG_PCClientPCREventStruc.event field.
3. Place the resulting hash H1 into the TCG_PCClientPCREventStruc.digest field.
4. Perform a TPM_Extend operation using H1 as input to the TPM_Extend.

5. Place the resulting TCG_PCClientPCREventStruc into the Measurement Log.
6. Increment the event counter and place it into the output parameter block.
7. Place H1 into the output parameter block.
8. Return.

If input parameter HashDataPtr and input parameter HashDataLen are both NULL, this function MUST perform the following:

1. Treat the input parameter LogDataPtr as a completed TCG_PCClientPCREventStruc including the TCG_PCClientPCREventStruc.digest field.
2. Perform a TPM_Extend operation using the TCG_PCClientPCREventStruc.digest field as input to the TPM_Extend.
3. Place the resulting TCG_PCClientPCREventStruc into the Measurement Log.
4. Increment the event counter and place it into the output parameter block.
5. Place TCG_PCClientPCREventStruc.digest field into the output parameter block.
6. Return.

On entry:

(AH)	=	BBh
(AL)	=	01h
(ES)	=	Segment portion of the pointer to the HashLogExtendEvent input parameter block
(DI)	=	Offset portion of the pointer to the HashLogExtendEvent input parameter block
(DS)	=	Segment portion of the pointer to the HashLogExtendEvent output parameter block
(SI)	=	Offset portion of the pointer to the HashLogExtendEvent output parameter block
(EBX)	=	41504354h
(ECX)	=	0
(EDX)	=	0

On return:

(EAX)	=	Return Code as defined in Section 12.3
(DS:SI)	=	Referenced buffer updated to provide return results

All other registers are preserved.

12.6.1 TCG_HashLogExtendEvent Input Parameter Block

For backward compatibility, two formats of the Input Parameter Block are allowed. The BIOS differentiates between them by examining the IPBLength field, which is set to specified lengths for each format.

12.6.1.1 TCG_HashLogExtendEvent Input Parameter Block Format 1

Table 14: TCG_HashLogExtendEvent Input Parameter Block Format 1

Offset	Size	Field Name	Description
00h	WORD	IPBLength	The length, in bytes of the input parameter block, set to 0018h.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	DWORD	HashDataPtr	The 32-bit physical address of the start of the data buffer to be hashed, extended, and logged.
08h	DWORD	HashDataLen	The length, in bytes, of the buffer referenced by HashDataPtr.
0Ch	DWORD	PCRIndex	The PCR number to which the hashed result is to be extended. This value SHOULD match TCG_PCClientPCREventStruc.pcrIndex. If these values are different, the BIOS SHOULD return an error.
10h	DWORD	LogDataPtr	The 32-bit physical address of the start of the data buffer containing the TCG_PCClientPCREventStruc data structure.
14h	DWORD	LogDataLen	The length, in bytes, of the TCG_PCClientPCREventStruc data structure.

12.6.1.2 TCG_HashLogExtendEvent Input Parameter Block Format 2

Table 15: TCG_HashLogExtendEvent Input Parameter Block Format 2

Offset	Size	Field Name	Description
00h	WORD	IPBLength	Same as Format 1 (see Table 14)
02h	WORD	Reserved	Same as Format 1 (see Table 14)
04h	DWORD	HashDataPtr	Same as Format 1 (see Table 14)
08h	DWORD	HashDataLen	Same as Format 1 (see Table 14)
0Ch	DWORD	PCRIndex	Same as Format 1 (see Table 14)
10h	DWORD	Reserved	Reserved for compatibility
14h	DWORD	LogDataPtr	Same as Format 1 (see Table 14)
18h	DWORD	LogDataLen	Same as Format 1 (see Table 14)

12.6.2 TCG_HashLogExtendEvent Output Parameter Block

There is only one format for the Output Parameter Block.

Table 16: TCG_HashLogExtendEvent Output Parameter Block

Offset	Size	Field Name	Description
00h	WORD	OPBLength	The length, in bytes, of the output parameter block. If hash algorithm is SHA-1, which has a 20h byte output, this value will be set by the INT 1Ah interface 001Ch.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	DWORD	EventNumber	The event number of the event just logged.
08h	Depends on hash function	HashValue	The TPM_HASH result of the HashAll function. Since the *TCG Main Specification* states this hash function uses SHA-1, the size of this field will be 20h bytes.

12.7 TCG_PassThroughToTPM

INT 1Ah, (AH)=BBh, (AL)=02h

This function provides a pass-through capability from the caller to the system's TPM. The caller is responsible for building the command byte-stream to be sent and interpreting the resulting byte-stream returned. These are defined in the *TPM Main Specification.*

The caller should verify the availability of this function by issuing a previous call to the TCG_StatusCheck function; that way the caller can be assured that calls to this function preserve the register contents (including the upper 16 bits of 32-bit registers).

The TPM in and out Operands are defined in the *TCG Main Specification*.

On entry:

(AH)	=	BBh
(AL)	=	02h
(ES)	=	Segment portion of the pointer to the TPM input parameter block
(DI)	=	Offset portion of the pointer to the TPM input parameter block
(DS)	=	Segment portion of the pointer to the TPM output parameter block
(SI)	=	Offset portion of the pointer to the TPM output parameter block
(EBX)	=	41504354h
(ECX)	=	0
(EDX)	=	0

On return:

(EAX)	=	Return Code as defined in Section 12.3
(DS:SI)	=	Referenced buffer updated to provide return results

All other registers are preserved.

12.7.1 TCG_PassThroughToTPM Input Parameter Block

Table 17: TCG_PassThroughToTPM Input Parameter Block

Offset	Size	Field Name	Description
00h	WORD	IPBLength	The length, in bytes, of the input parameter block set to 0008h plus the size of the TPMOperandIn.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	WORD	OPBLength	Size of TPM Output Parameter Block allocated. BIOS will return error ??? if OPBLength is is less than 4 bytes.
06h	WORD	Reserved	
08h	BYTE	TPMOperandIn	The TPM Operand Parameter Block to send to the TPM.

12.7.2 TCG_PassThroughToTPM Output Parameter Block

Table 18: TCG_PassThroughToTPM Output Parameter Block

Offset	Size	Field Name	Description
00h	WORD	OPBLength	The length, in bytes, of the output parameter block, set by the INT 1Ah interface to 0004h plus the size of the TPMOperandOut.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	BYTE	TPMOperandOut	The TPM Operand Parameter Block received from the TPM.

12.8 TCG_ShutdownPreBootInterface

INT 1Ah, (AH)=BBh, (AL)=03h

The IPL Code issues this call once it has its runtime access to the TPM available, and causes the system firmware to no longer respond to TCG BIOS requests through this interface until the next system restart.

Upon completion of this function, all BIOS functions defined in this specification, except the function TCG_StatusCheck, MUST perform no action and return the error code TPM_INTERFACE_SHUTDOWN. The function TCG_StatusCheck SHOULD return all defined registers as defined but with the error code TPM_INTERFACE_SHUTDOWN.

Calling this function is optional.

On entry:

(AH)	=	BBh
(AL)	=	03h
(EBX)	=	41504354h

On return:

(EAX)	=	Return Code as defined in Section 12.3

All other registers are preserved.

12.9 TCG_HashLogEvent

INT 1Ah, (AH)=BBh, (AL)=04h

This function performs the same function as the TCG_HashLogExtendEvent function except it does not perform the TPM_Extend operation.

Apply the same conditions and perform the same operations as described in Section 12.6 under "Description of Process" except this function MUST NOT perform the TPM_Extend operation.

There are two reasons for calling this function:

1. To create an event associated with a previous extend operation. The extend operation might have been done within an environment (e.g., by the CRTM) were no memory might be available to create the measurement log entry. Using this function, the BIOS can create the associated log entry. In this case, the caller MUST set the PCRIndex field to the PCR into which the extend was performed.
2. An informative event entry for which no extend operation occurred. The values within this entry cannot be verified; rather they may serve an informative or delimiting function. In this case, the EventType should provide an indication of this; but since the PCRIndex field still exists within the event structure, the caller SHOULD set to the PCRIndex to -1 (all ones) to provide a further indication that no extend was performed.

On entry:

(AH)	=	BBh
(AL)	=	04h
(ES)	=	Segment portion of the pointer to the LogEvent input parameter block
(DI)	=	Offset portion of the pointer to the LogEvent input parameter block
(DS)	=	Segment portion of the pointer to the LogEvent output parameter block
(SI)	=	Offset portion of the pointer to the LogEvent output parameter block
(EBX)	=	41504354h
(ECX)	=	0
(EDX)	=	0

On return:

(EAX)	=	Return Code as defined in Section 12.3
(DS:SI)	=	Referenced buffer updated to provide return results

All other registers are preserved.

12.9.1 TCG_HashLogEvent Input Parameter Block

Table 19: TCG_HashLogEvent Input Parameter Block

Offset	Size	Field Name	Description
00h	WORD	IPBLength	The length, in bytes,of the input parameter block, set to 001Ch.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	DWORD	HashDataPtr	The 32-bit physical address of the start of the data buffer to be logged.
08h	DWORD	HashDataLen	The length, in bytes, of the buffer referenced by HashDataPtr.
0Ch	DWORD	PCRIndex	The PCR number associated with this event. This value SHOULD match TCG_PCClientPCREventStruc.pcrIndex. If these values are different, the BIOS SHOULD return an error.
10h	DWORD	LogEventType	This value SHOULD match the value in TCG_PCClientPCREventStruc.eventType. If these values are different, the BIOS SHOULD return an error.
14h	DWORD	LogDataPtr	The 32-bit physical address of the start of the data buffer containing the TCG_PCClientPCREventStruc data structure.
18h	DWORD	LogDataLen	The length, in bytes, of the TCG_PCClientPCREventStruc data structure.

12.9.2 TCG_HashLogEvent Output Parameter Block

Table 20: TCG_HashLogEvent Output Parameter Block

Offset	Size	Field Name	Description
00h	WORD	OPBLength	The length, in bytes, of the output parameter block, set by the INT 1Ah interface to 0008h.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	DWORD	EventNumber	The event number of the event just logged.

12.10 TCG_HashAll

INT 1Ah, (AH)=BBh, (AL)=05h

This function performs the required hash operation on the input data and returns the resulting hash to the caller.

On entry:

(AH)	=	BBh
(AL)	=	05h
(ES)	=	Segment portion of the pointer to the HashAll input parameter block
(DI)	=	Offset portion of the pointer to the HashAll input parameter block
(DS)	=	Segment portion of the pointer to the Digest
(SI)	=	Offset portion of the pointer to the Digest
(EBX)	=	41504354h

(ECX) = 0

(EDX) = 0

On return:

(EAX) = Return Code as defined in Section 12.3

(DS:SI) = Referenced buffer updated to provide return results

All other registers are preserved.

12.10.1 TCG_HashAll Input Parameter Block

Table 21: TCG_HashAll Input Parameter Block

Offset	Size	Field Name	Description
00h	WORD	IPBLength	The length, in bytes, of the input parameter block, set to 0010h.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	DWORD	HashDataPtr	The 32-bit physical address of the start of the data buffer to be hashed.
08h	DWORD	HashDataLen	The length, in bytes, of the buffer referenced by HashDataPtr.
0Ch	DWORD	AlgorithmID	The algorithm to use. In TCG v1, this MUST be TPM_ALG_SHA.

12.11 TCG_TSS

INT 1Ah, (AH)=BBh, (AL)=06h

This function provides optional TSS capabilities. If any TSS commands are implemented through this function, TSS_GetCapability MUST be implemented to give the caller the ability to determine which TSS Operations are supported. If no TSS Operations are supported, this function MUST return with EAX = TCG_PC_UNSUPPORTED.

The TSS in and out Operands are defined in the *TCG TSS Specification.* The TSS Operand Parameter block is defined in a currently unreleased specification. Until the specification is released, this function SHOULD NOT be supported and SHOULD indicate so by returning the error code TCG_PC_UNSUPPORTED.

On entry:

(AH) = BBh

(AL) = 06h

(ES) = Segment portion of the pointer to the TSS input parameter block

(DI) = Offset portion of the pointer to the TSS input parameter block

(DS) = Segment portion of the pointer to the TSS output parameter block

(SI) = Offset portion of the pointer to the TSS output parameter block

(EBX) = 41504354h

(ECX) = 0
(EDX) = 0

On return:

(EAX) = Return Code as defined in Section 12.3
(DS:SI) = Referenced buffer updated to provide return results
All other registers are preserved.

12.11.1 TCG_TSS Input Parameter Block

Table 22: TCG_TSS Input Parameter Block

Offset	Size	Field Name	Description
00h	WORD	IPBLength	The length, in bytes, of the input parameter block, set to 0008h plus the size of the TSSOperandIn.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	WORD	OPBLength	Size of TSS Output Parameter Block allocated.
06h	WORD	Reserved	
08h	BYTE	TSSOperandIn	The TSS Operand Parameter Block to send to the TPM.

12.11.2 TCG_TSS Output Parameter Block

Table 23: TCG_TSS Output Parameter Block

Offset	Size	Field Name	Description
00h	WORD	OPBLength	The length, in bytes, of the output parameter block, set to 0004h plus the size of TSSOperandOut.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	BYTE	TSSOperandOut	The TSS Operand Parameter Block received from the TSS.

12.12 TCG_CompactHashLogExtendEvent

INT 1Ah, (AH)=BBh, (AL)=07h

This function performs hashing of the event or the event data, extends the event to a PCR, then places the resulting TCG_PCClientPCREventStruc into the event log, very similar to TCG_HashLogExtendEvent but with a simplified interface so it can be used without the setup of any memory structure. While useful by other Pre-Operating System State's components, this function's use is tailored for boot record code where code space is very limited.

This function MUST create an event log entry of the Event Type: EV_COMPACT_HASH. The value contained in ESI will be placed into the event field with the eventDataSize being set to 4. This function MUST not treat the event field as part of the measurement; therefore, references to the event field will be informative in nature because they are not part of the measurement that was extended. On success, it will return the event number resulting from this call.

If this function cannot create a Measurement Log Entry (e.g., the Measurement Log is full), this function MUST perform the TPM_Extend operation.

If both ES and DI are NULL, this function MUST return the TCG_INVALID_INPUT_PARA error.

On entry:

(AH)	=	BBh
(AL)	=	07h
(ES)	=	Segment portion of the pointer to the start of the data buffer to be hashed
(DI)	=	Offset portion of the pointer to the start of the data buffer to be hashed
(ESI)	=	The informative value to placed into the event field.
(EBX)	=	41504354h
(ECX)	=	The length, in bytes, of the buffer referenced by ES:DI
(EDX)	=	The PCR number (PCRIndex) to which the hashed result is to be extended

On return:

(EAX)	=	Return Code as defined in Section 12.3
(EDX)	=	Event number of the event that was logged

All other registers are preserved.

12.12.1 TCG_BIOSReserved

INT 1Ah, (AH)=BBh, (AL)=08h to 07Fh

Remaining subfunctions in the range 07h to 07Fh are reserved for future definition by this specification.

12.12.2 TCG_BIOSVendorReserved

INT 1Ah, (AH)=BBh, (AL)=80h to 0FFh

Reserved for Vendor specific functions.

On entry:

(AH)	=	BBh
(AL)	=	nnh
(EBX)	=	41504354h

13. Error and Return Codes

Table 24 lists the return codes used for both the INT 1Ah interface and the MA / MP interfaces (if implemented).

The catalog of error and return codes can be extended to include TPM vendor specific return codes at the end of the list.

If either driver fails to communicate with the TPM, it MUST do one of the following:

1. Permanently disable the connection to the TPM
2. Take action to prevent the Host Platform from loading the operating system
3. Perform a Host Platform Reset
4. Force transfer control of the Host Platform to a manufacturer approved environment

Table 24: INT 1Ah and MA / MP Interface Error and Return Codes

Error Code	Value	Description
TCG_OK	00h	Indicator of successful execution of the function.
TPM_RET_BASE	01h	Base of return codes.
TCG_GENERAL_ERROR	TPM_RET_BASE + 00h	A general unidentified error occurred.
TCG_TPM_IS_LOCKED	TPM_RET_BASE + 01h	The access cannot be granted the device is open.
TCG_NO_RESPONSE	TPM_RET_BASE + 02h	No response from the TPM device.
TCG_INVALID_RESPONSE	TPM_RET_BASE + 03h	The response from the TPM was invalid.
TCG_INVALID_ACCESS_REQUEST	TPM_RET_BASE + 04h	The access parameters for this function are invalid.
TCG_FIRMWARE_ERROR	TPM_RET_BASE + 05h	Firmware error during start up.
TCG_INTEGRITY_CHECK_FAILED	TPM_RET_BASE + 06h	Integrity checks of TPM parameter failed.
TCG_INVALID_DEVICE_ID	TPM_RET_BASE + 07h	The device ID for the TPM is invalid.
TCG_INVALID_VENDOR_ID	TPM_RET_BASE + 08h	The vendor ID for the TPM is invalid.
TCG_UNABLE_TO_OPEN	TPM_RET_BASE + 09h	Unable to open a connection to the TPM device.
TCG_UNABLE_TO_CLOSE	TPM_RET_BASE + 0Ah	Unable to close a connection to the TPM device.
TCG_RESPONSE_TIMEOUT	TPM_RET_BASE + 0Bh	Time out for TPM response.
TCG_INVALID_COM_REQUEST	TPM_RET_BASE + 0Ch	The parameters for the communication access are invalid.
TCG_INVALID_ADR_REQUEST	TPM_RET_BASE + 0Dh	The address parameter for the access is invalid.
TCG_WRITE_BYTE_ERROR	TPM_RET_BASE + 0Eh	Bytes write error on the interface.
TCG_READ_BYTE_ERROR	TPM_RET_BASE + 0Fh	Bytes read error on the interface.
TCG_BLOCK_WRITE_TIMEOUT	TPM_RET_BASE + 10h	Blocks write error on the interface.
TCG_CHAR_WRITE_TIMEOUT	TPM_RET_BASE + 11h	Bytes write time out on the interface.
TCG_CHAR_READ_TIMEOUT	TPM_RET_BASE + 12h	Bytes read time out on the interface.
TCG_BLOCK_READ_TIMEOUT	TPM_RET_BASE + 13h	Blocks read error on the interface.
TCG_TRANSFER_ABORT	TPM_RET_BASE + 14h	Transfer abort in communication with TPM device.
TCG_INVALID_DRV_FUNCTION	TPM_RET_BASE + 15h	Function number (AL-Register) invalid for this driver.
TCG_OUTPUT_BUFFER_TOO_SHORT	TPM_RET_BASE + 16h	Output buffer for the TPM response to short.
TCG_FATAL_COM_ERROR	TPM_RET_BASE + 17h	Fatal error in TPM communication.
TCG_INVALID_INPUT_PARA	TPM_RET_BASE + 18h	Input parameter for the function invalid.

Error Code	Value	Description
TCG_TCG_COMMAND_ERROR	TPM_RET_BASE + 19h	Error during execution of a TCG command.
TCG_Reserved1	TPM_RET_BASE + 1Ah	
TCG_Reserved2	TPM_RET_BASE + 1Bh	
TCG_Reserved3	TPM_RET_BASE + 1Ch	
TCG_Reserved4	TPM_RET_BASE + 1Dh	
TCG_Reserved5	TPM_RET_BASE + 1Eh	
TCG_Reserved6	TPM_RET_BASE + 1Fh	
TCG_INTERFACE_SHUTDOWN	TPM_RET_BASE + 20h	TPM BIOS interface has been shutdown using the TCG_ShutdownPreBootInterface.
TCG_PC_UNSUPPORTED	TPM_RET_BASE + 21h	The requested function is not supported.
TCG_PC_TPM_NOT_PRESENT	TPM_RET_BASE + 22h	The TPM is not installed.
TCG_PC_TPM_DEACTIVATED	TPM_RET_BASE + 23h	The TPM is deactivated.
TCG_VENDOR_BASE_RET	80h	Start point for return codes are reserved for use by TPM vendors.

14. TPM Driver Interfaces

Start of informative comment

The *Version 1.1 TPM Specification* and the *PC Client Specification* did not specify an interface protocol. This made it difficult for BIOS implementers to create a “standard” BIOS level TPM driver interface that was compatible with all TPMs. Host Platform manufacturers wanted to be able to select TPMs without changing their BIOS, so the MA and MP interfaces were created to provide the necessary common interface.

However, this version of the *PC Client Specification* (1.2) includes an interface specification that standardizes the TPM’s interface to the driver. The operating systems will also standardize to this interface. This reduces and, perhaps, eliminates the need for a driver-to-BIOS interface such as the MA and MP drivers.

There may be some situations where maintaining a standardized separation of functions is required. In these cases, the driver-to-BIOS interface is still useful. Therefore, it is not deleted and is still maintained in this version of the *PC Client Specification*. However, is should be considered deprecated and may not be continued in future versions of the *PC Client Specification*.

End of informative comment

1. The TPM manufacturer (or other entity supplying a TPM to an OEM) MAY provide the drivers (MA or MP) specified in this section.
2. The OEM MAY require the TPM manufacturer (or other entity supplying a TPM to an OEM) to provide the drivers (MA or MP) specified in this section.
3. The decision to supply or require the drivers (MA or MP) specified in this section is entirely a business decision on the part of the supplier or requestor and SHALL NOT alter any qualifications to make a claim of compliance to any TCG specification.

14.1 Module Architectures

14.1.1 TPM Supplied BIOS Drivers

Start of informative comment

The TPM vendor may supply one or two BIOS drivers in addition to the normal operating system drivers depending on the type of BIOS. One of these is the Memory Present (MP) driver for the Post-BIOS environment. It supports the INT 1Ah TPM-Communication-Interface defined in this specification. The second BIOS driver is the Memory Absent (MA) driver, which will run in a memory-less and stack-less environment. Typically this will be in the BIOS Boot Block if it is a Compound BIOS.

Both the MA and the MP driver will be provided for a BIOS with the Compound BIOS architecture while only the MP driver will be provided for a BIOS with the Integrated BIOS architecture.

This interface is normative for the purpose of providing a standard interface to allow interoperability for both TPM and Host Platform manufacturers. However, the interfacing modules (i.e., the BIOS and the driver) are neither general-purpose nor widely distributed, rather each are tightly controlled by their respective owners. It is a desired, but likely not a globally achievable, goal that all BIOSes interface to TPM drivers without some level of cooperation between the two module developers.

End of informative comment

14.1.2 Object Format of BIOS Drivers

Both the MA and MP drivers provide a standard object format to the BIOS vendor as described in this section. Table 25 in Section 14.1.5.2 describes what the header of the BIOS drivers will look like and where the driver code should start.

14.1.3 Driver and TPM errors

Start of informative comment

The PCRs are used to begin and continue the chain of trust. To be valid, this chain must begin at the root of trust of measurement which is the CRTM. If the CRTM cannot make the required initial measurements, untrusted components executing after the CRTM completes execution can extend values into the PCRs from the state they are in at Host Platform Reset allowing untrusted software to masquerade as trusted software. Therefore, if the CRTM cannot make the initial measurements, it must prevent all further communication to the TPM.

End of informative comment

In the case of a fatal driver error (either Memory Absent or Memory Present), the TPM communication channel or the TPM MUST be closed for the duration of the Host Platform's boot cycle including prevention of access to the TPM by any operating system component, driver, or application.

There is no required method for achieving this. Example methods include, but are not limited to, performing a CRTM or BIOS controlled Host Platform Reset, performing a CPU halt instruction, or issuing a command to a port that stops communication to the TPM until a Host Platform Reset occurs. It is generally expected that these actions will be taken by the BIOS since the driver can return timeout or other communication errors but, again, the implementation is Host Platform specific.

14.1.4 Locality Access

The BIOS MUST access the TPM using Locality 0. The BIOS SHOULD access the TPM using the Locality 0 memory-mapped addresses but MAY use any of the legacy I/O addresses.

14.1.5 BIOS Driver Header

14.1.5.1 BIOS Driver Header Location

The location of the BIOS Driver Header MUST be at the beginning of the binary image for both the MA and the MP drivers.

14.1.5.2 BIOS Driver Header Format

Table 25: BIOS Driver Header Format

Offset	Size	Default-Value	Description
00h	WORD	55AAh	Signature used to designate the start of the BIOS driver. This is deliberately set different than the Option ROM header.
02h	DWORD		Pointer to beginning of code (offset to entry point for the driver). This is a 32-bit near entry point.
06h	WORD		Total size of the driver in bytes (including the header).
08h	DWORD	FED40000h	Base Address. The driver MUST attempt to access the TPM using this address. If access to the TPM fails using this address, the driver MAY use the Alternate Address. The driver MUST fail this request if any request is made to any Locality 1-4 addresses. 1. If this location is FED40000h (default), the driver MUST use the Locality 0 memory-mapped addresses to access the TPM as described in the TIS. The driver MUST use the locality protocol as specified in the TIS. 2. If this location is not FED40000h (i.e., not default) it MUST indicate one of the legacy I/O addresses as described in TIS Section 8.4. The driver does not use the Locality protocol because that protocol is not used for legacy I/O addressing of the TPM.
0Ch	DWORD	00000000h	Alternate Address. This field is used to indicate an alternate address. The same rules apply as specified in the Base Address except if this fails no further attempt is made to access the TPM. That is, do not try Base Address and cause an endless loop. The value "0" in this field indicates no alternate address is specified and only the Base Address may be attempted.
10h	BYTE	FFh	IRQ Level (00h is not assigned; FFh is not required). This field is not relevant to and MUST be ignored by the MA and MP drivers.
11h	BYTE	FFh	DMA Channel (FFh is not assigned) (as set by BIOS). This field is not relevant to and MUST be ignored by the MA and MP drivers.
12h	BYTE		XOR-Checksum of entire driver including this header at driver build time. This is not maintained by the BIOS.
13h	BYTE	00h	Reserved and set to "0".
14h	DWORD	00000000h	PCI PFA if appropriate. Not used in TPM family 1.2.
18h	DWORD	00000000h	USB, CardBus, etc. Not used in TPM family 1.2.
1Ch	DWORD		Location of TPM Configuration port.
20h	Variable		Reserved for vendor specific data . If this vendor specific data area is not used, this begins the entry point into the driver.[9]
XXh			Entry point into driver.

[9] The memory area that is the entry point into the driver is not technically part of the header, rather the driver itself. These offsets and their entries in this table are provided for the convenience of the reader.

14.1.6 Basic Assumptions for Both BIOS Drivers

14.1.6.1 CMOS Timer

The CMOS Real Time Clock (RTC) will be available for both drivers and initialized by the caller. The RTC will be available by its legacy I/O addresses.

14.1.6.2 Motherboard Initialization

All motherboard chipset initialization (concerning the communication channel to TPM device) will be completed by the CRTM or POST-BIOS prior to calling the MA or MP driver.

14.1.6.3 Basic Requirements

The BIOS drivers MUST fulfill the following requirements:

1. The drivers MUST be completely self-contained since no BIOS services should be used.
2. The drivers MUST check the validity of all the input parameters.
3. The drivers MUST include block chaining for the transmission of large data blocks to and from the TPM device.
4. The drivers are responsible to add and remove all TPM-Vendor specific protocol information to the TCG-Transfer-Data (TCG-Command).

14.2 Memory Absent (MA) Driver

14.2.1 Architecture

Start of informative comment

This driver is designed to operate in a very limited environment. Specifically, it operates without memory, using only the CPU registers for data storage. The driver MUST be completely self-contained since no BIOS services will be available.

It is expected to be used in the BIOS Boot Block of Compound BIOSes. It is not required for motherboards containing an Integrated BIOS to implement this driver; instead they may choose to implement only the Memory Present (MP) driver described in Section 14.3.

The purpose of the MA driver is to hash and extend the first portion of BIOS code before jumping to that code. The MA driver and TPM MUST perform the hash and extend operation in less than 2 seconds.

End of informative comment

14.2.2 MA Driver Memory Model

In the Memory Absent driver, the memory model MUST be “Big-Real”. The MA driver code MUST be 16 bit, “Big-Real” code. The MA driver MUST be paragraph aligned. The code MUST NOT make any assumption about the location of the MA driver because the BIOS may locate the MA driver at any valid address.

Calls into the MA driver and returns from the MA driver MUST be 16-bit far.

14.2.3 MA Driver Segments

In the Memory Absent driver, the data segment SHALL be a flat. (i.e., 0-based with 4-GB limit for DS and ES data segments). Table 26 lists the segments and their requirements.

Table 26: MA Driver DS and ES Data Segments

Register	Description
CS	Normal "real-mode" code segment with no "protected-mode" artifacts. Upon entry to the MA driver, this MUST based at the beginning to the BIOS Driver Header.
DS	0-based with 4 GB limit.
ES	0-based with 4 GB limit.
FS	The MA driver MUST NOT assume anything about this register. The MA driver MUST preserve the value and MUST NOT change the limit.
GS	The MA driver MUST NOT assume anything about this register. The MA driver MUST preserve the value and MUST NOT change the limit.
SS	Normal "real-mode" stack segment with no "protected-mode" artifacts. The MA driver MUST preserve the value and MUST not change the limit.

14.2.4 MA Driver Limitations

1. No DMA
2. No IRQ
3. No Physical Memory
4. MA driver register (General-Purpose and Segment register) usage. See Table 27.

Table 27: MA Driver Register (General-Purpose and Segment Register) Usage

Register	Size	In / Out	Description
EAX	32	Not available	Driver must preserve this register.
EBX	32	Not available	Driver must preserve this register.
ECX	32	In / Out	Driver I/O; set by the caller.
EDX	32	In / Out	Driver I/O; set by the caller.
ESI	32	Not available	Driver must preserve this register.
EDI	32	Not available	Driver must preserve this register.
ESP	32	In (Offset)	Offset of the pointer to argument packet; see Section 14.2.5. Set by the caller.
CS	16	In: Set by BIOS Out:Preserved	BIOS sets this value; driver preserves value.
DS	16	In: Set by BIOS Out: Preserved	BIOS sets this value; driver preserves value.
ES	16	In: Set by BIOS Out: Preserved	BIOS sets this value;, driver preserves value.
FS	16	In: Set by BIOS Out: Preserved	BIOS sets this value; driver preserves value.
GS	16	In: Set by BIOS Out: Preserved	BIOS sets this value; driver preserves value.

Register	Size	In / Out	Description
SS	16	In (Segment)	Segment of the pointer to argument packet see Section 14.2.5. Set by the caller.

5. All other registers MAY be used as working registers by the MA driver without preserving them.
6. The IA-32 processor (PIII, Athlon, or equivalent processor) architecture supports MMX/3DNow and FPU. It MAY be negotiated between the BIOS vendor (more specifically the vendor of the CRTM) and the supplier of the CRTM driver (typically the TPM vendor) that this driver can use the MMX/3DNow register MM0 through MM7 as working registers as well as the MMX instructions. If the driver uses any other registers, that usage MUST be documented. (Note: The MMX registers are mapped to the physical location of the floating-point registers (R0 through R7). This means when a value is written into an MMX register using an MMX instruction, the value also appears in the corresponding floating-point register.)

14.2.5 MA Driver Argument Packet Structure

On entry to the MA driver, SS:ESP points to an instance of this structure. The CRTM MAY have one or more of these structures per function to allow multiple calls into a single function from different locations.

```
MADriverArgPacketStruct      STRUC
   ReturnAddr    DD ?  ; [IN] Return address. Allows driver to retrun via RETF.
   HeaderPtr     DD ?  ; [IN] Pointer to the BIOS Driver Header (See
                                 Section 14.1.2).
   FunctionNum   DB ?  ; [IN] Function number identifing the function to perform.
MADriverArgPacketStruct      ENDS
```

2540 ### 14.2.6 Parameters and Structures

14.2.6.1 Parameter pbInBuf

BYTE **pbInBuf*	
Description	Pointer to the start address of the input data for the data transfers to the TPM.

14.2.6.2 Parameter dwInPCRLen

DWORD *dwInPCRLen*	
Description	Upper 16 bits contain the PCRIndex. The lower 16 bits contain the length of the input data record – 1. (i.e., FFFFh hashes 65536 bytes).

14.2.6.3 Parameter bMAInitTPMFctId

BYTE *bMAInitTPMFctId*	
Description	Selects the TPM Operation for the CRTM driver initialization. 00h = No TPM-Operation is selected. To activate the TPM_Startup command set this parameter with a TPM_STARTUP_TYPE identifier specified in the *TCG Main Specification* (see TPM_Startup section in *TCG Main Specification*).

14.2.6.4 Parameter bMAPhyPresenceTPMCmdId

WORD *bMAPhyPresenceTPMCmdId*	
Description	Selects the TPM Operation for the PhysicalPresence command. This value is used in the TPM-Param-Block of the TPM_PhysicalPresence command. For the detailed definition of this identifier, see the *TCG Main Specification.*

14.2.7 MA Driver Function Interface

The function number is contained in the FunctionNum field of the MADriverArgPacketStruct structure (Refer to Section 14.2.5). The base for the function numbers is 01h. The offset for vendor specific driver function numbers is 80h. All functions return their exit code in the DL register.

14.2.7.1 Function MAInitTPM (Function Number: 01h)

The first call to the MA driver must execute this function. This function does the initialization of the TPM and establishes and verifies the communication (with the parameters from the header) between the MA driver and the TPM. If a TPM operation is selected by the *bTPMInitCRTMFctId* parameter, this function will send the command string to the TPM.

A TPM device can be opened with the same address only once by one host at a time. If the requested access cannot be granted (e.g., invalid input parameter) or if opening the connection to the TPM ends unsuccessfully, the function returns corresponding *errorCode*.

BYTE MAInitTPM (**BYTE** *bMAInitTPMFctId*);	
Input Parameters	*DL = bMAInitTPMFctId* Function identifier for the TPM_Startup operation (see Section 14.2.6.3).
Return Value	*DL = return value of this function* One of the following values: **TPM_OK** **TPM_IS_LOCKED** **TPM_NO_RESPONSE** **TPM_INVALID_RESPONSE** **TPM_RESPONSE_TIMEOUT** **TPM_INVALID_ACCESS_REQUEST** **TPM_FIRMWARE_ERROR** **TPM_GENERAL_ERROR** **TPM_TRANSFER_ABORT** **TPM_TCG_COMMAND_ERROR**

14.2.7.2 Function MAHashAllExtendTPM (Function Number: 02h)

Start of informative comment

This function sends TPM hash operations to the TPM to hash the specified memory range. This function performs TPM_SHA1Start, TPM_SHA1Update, and TPM_SHA1CompleteExtend to the PCR specified in *dwInPCRLen* parameter.

It transmits the data from the input buffer (**pbInBuf*) to the TPM and reads the response from the TPM. After successful Power-On and opening a TPM connection, the host can use this function to measure the POST BIOS. This function is responsible for any byte stream buffering and error handling during the interaction with the TPM device over the communication interface. All vendor specific transport protocol information are added and removed by this function.

If no open connection to a TPM device is available, if this function receives no valid response from the TPM, if the function calling parameters are invalid, or the transmission of the data block to the TPM ends unsuccessfully, the function fails and returns corresponding *errorCode*.

Note: Only a 16-bit value is proved to indicate data length. This means if the size of the data is larger than this value can represent multiple calls to this function will be required. Note further that each call results in a complete hash and extend so if multiple calls are needed they will appear as multiple extend operations not a single hash and extend.

End of informative comment

BYTE MAHashAllExtendTPM (**DWORD** **pbInBuf*, **DWORD** *dwInPCRLen*);	
Input Parameters	*EDX =*pbInBuf* Pointer to the start address of input buffer containing the data for the TPM device (see Section 14.2.6.1). *ECX = dwInPCRLen* PCRIndex and Length of the input buffer data (see Section 14.2.6.2).
Return Value	*DL = return value of this function* One of the following values: **TPM_OK** **TPM_IS_LOCKED** **TPM_NO_RESPONSE** **TPM_INVALID_RESPONSE** **TPM_RESPONSE_TIMEOUT** **TPM_INVALID_ACCESS_REQUEST** **TPM_FIRMWARE_ERROR** **TPM_GENERAL_ERROR** **TPM_TRANSFER_ABORT** **TPM_TCG_COMMAND_ERROR**

14.2.7.3 Function MAPhysicalPresenceTPM (Function Number: 03h)

Start of informative comment

This function sends the TSC_PhysicalPresence operations with the command value specified in the *bMAPhyPresenceTPMCmdId* parameter to the TPM.

If no open connection to a TPM device is available, if this function receives no valid response from the TPM, if the function calling parameters are invalid, or the transmission of the data block to the TPM ends unsuccessfully, the function fails and returns corresponding *errorCode*.

End of informative comment

BYTE MAPhysicalPresenceTPM (**WORD** *bMAPhyPresenceTPMCmdId*);	
Input Parameters	*DX = bMAPhyPresenceTPMCmdId* Command identifier for the TPM_PhysicalPresence operation (see Section 14.2.6.4).
Return Value	*DX = return value of this function* One of the following values: **TPM_OK** **TPM_IS_LOCKED** **TPM_NO_RESPONSE** **TPM_INVALID_RESPONSE** **TPM_RESPONSE_TIMEOUT** **TPM_INVALID_ACCESS_REQUEST** **TPM_FIRMWARE_ERROR** **TPM_GENERAL_ERROR** **TPM_TRANSFER_ABORT** **TPM_TCG_COMMAND_ERROR**

14.3 Memory Present (MP) Driver

14.3.1 Architecture

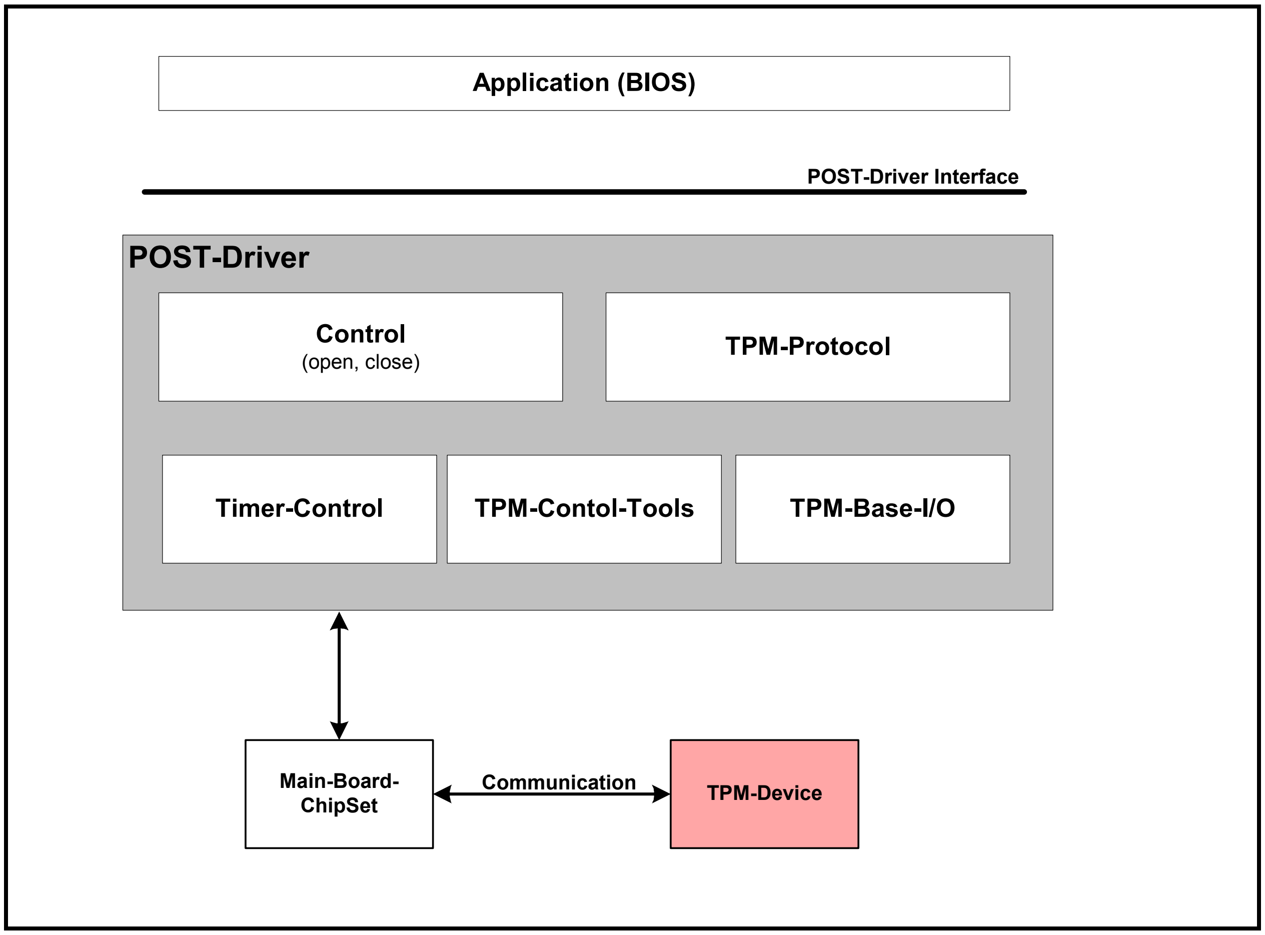
Start of informative comment

The MP driver is a module of the TCG software for the TPM device. The main goal for the MP driver is to support the customer in their BIOS integration of the TPM control and communication software. The driver also includes functions for the handling of the data block transmission protocol between the TPM device and the host system.

As discussed above, the POST driver will need to be 32-bit relocatable code. The BIOS code will bring up memory, load the POST driver, and call the start of the POST driver. Prior to calling the MP driver, the BIOS will set a base address for the TPM. This base address will be stored as part of the driver header. All of the configuration data (not JUST the base address) will be set by the BIOS prior to calling the MP driver.

All the data transfers from and to the TPM are done through this module. Through this module, a Host system reads, writes, and controls the TPM. The application and MP driver communicate through the chipset interface with the TPM device.

End of informative comment



2605 **Figure 7: Pre-Operating System State Driver Interface**

14.3.2 MP Driver Memory Model

In the Memory Present driver, the memory model MUST be flat 32-bit protected-mode. The code is position independent code and MUST make no assumptions about the value of EIP.

Calls into the MP driver and returns from the MP driver MUST be 32-bit near.

14.3.3 MP Driver Segments

In the MP driver, the data segment SHALL be a flat, 32-bit address. This means that all descriptors referenced, including CS, MUST be 0-based flat descriptors. There is no requirement for the segment registers to be any particular value other than to reference a valid descriptor described in this subsection.

Table 28: MP Driver Segments

Register	Description
CS	Normal 32-bit flat segment and MUST allow data access.
DS	0-based with 4-GB limit.
ES	0-based with 4-GB limit.
FS	The MA driver MUST NOT assume anything about this register. The MA driver MUST preserve the value and MUST not change the limit.
GS	The MA driver MUST NOT assume anything about this register. The MA driver MUST preserve the value and MUST not change the limit.
SS	The stack is a 32-bit stack. The driver MUST document the required amount of stack required.

14.3.4 MP Driver Limitations

1. No Interrupts are allowed. The MP driver MUST poll the TPM.
2. The MP driver MAY be relocated after MAInitTPM and at any time between call MP driver functions.
3. The MP driver needs to be placed into ACPI non-reclaimable area. The driver MUST support being relocated between calls.
4. The resources allocated to the TPM MAY be changed by the BIOS between calling MP driver functions; therefore, the MAInitTPM function MUST be recallable.
5. All registers not used for return parameters MUST be preserved.
6. The MP driver needs to be built such that it has any data memory it requires is part of the body of the driver image.
7. If there is any paging, it MUST identity mapped. That is, virtual equals physical.
8. Any Host Processor feature that restricts execution by page MUST be disabled.

14.3.5 Parameters and Structures

14.3.5.1 Parameter pbInBuf

BYTE **pbInBuf*	
Description	Pointer to input data for the data transfers to the TPM.

14.3.5.2 Parameter pbOutBuf

BYTE **pbOutBuf*	
Description	Pointer to output buffer for the data transfers from the TPM.

14.3.5.3 Parameter dwInLen

DWORD *dwInLen*	
Description	Length of the input data record.

14.3.5.4 Parameter dwOutLen

DWORD *dwOutLen*	
Description	DWORD to store the length information of the output data record.

14.3.5.5 Structure TPMTransmitEntry

Start of informative comment

This structure is used by the TPMTransmit function to transfer the input and output parameters. The two output parameters (pbOutBuf, pdwOutLen) can be NULL-Pointers if no response is necessary or it has no meaning for the caller. This mode can be also helpful in memory less environments (e.g., BIOS-Boot-Block).

End of informative comment

```
TPMTransmitEntryStruct     STRUC
   pbInBuf      DD ?  ; [IN]       Pointer to input data for the data transfers to TPM.
   dwInLen      DD ?  ; [IN]       Length of the input data record.
   pbOutBuf     DD 0  ; [OUT]      Pointer to output buffer for the data from the TPM.
   dwOutLen     DD 0  ; [IN/OUT]   DWORD to store the length info of the output data
                                   record.
TPMTransmitEntryStruct     ENDS
```

The parameter pdwOutLen is both an input and an output parameter.

As input (entry point of this function) it specifies the maximum number of bytes, which can be read from the TPM device to the output buffer. If the function terminates successfully, the value of this variable is adjusted to match with the number of bytes received from the TPM.

14.3.5.6 Parameter lpTPMTransInfo

TPMTransmitEntryStruct *lpTPMTransInfo	
Description	Pointer to a TPMTransmitEntryStruct, which carries the input and output parameters for data transfer between Host System and the TPM device.

14.3.6 MP Driver Function Interface

The AL-register contains the function selector number for the different functions of this driver (the base for this is 01h). The offset for vendor specific driver function numbers is 80h. All these functions return their exit codes in the AL-register.

14.3.6.1 Function MPInitTPM (Function-Nr-AL-Register: 01h)

This function is performed the first time the driver is called. It is used to initialize the TPM if not already done by the BIOS Boot Block or if there are some differences between the communication parameters for the CRTM and POST Phase. This function must be also called if the BIOS moves the I/O address used by the TPM (such as if BIOS performs PnP conflict resolution).

This function does the initialization of the TPM and the driver and establishes (opens a connection) and verifies the communication (with the parameters from the header) between the POST driver and the TPM.

A TPM device can be opened with the same address only once by one host at a time. If the requested access cannot be granted (e. g., invalid input parameter) or if opening the connection to the TPM ends unsuccessfully, the function returns corresponding *errorCode*.

BYTE MPInitTPM (void);	
Input Parameters	*All necessary inputs are located in the driver header structure* (see Section 14.1.2).
Output Parameters	*None*
Return Value	*AL = return value of this function* One of the following values: **TPM_OK** **TPM_INVALID_ADR_REQUEST** **TPM_IS_LOCKED** **TPM_INVALID_DEVICE_ID** **TPM_INVALID_VENDOR_ID** **TPM_RESERVED_REG_INVALID** **TPM_FIRMWARE_ERROR** **TPM_UNABLE_TO_OPEN** **TPM_GENERAL_ERROR**

14.3.6.2 Function MPCloseTPM (Function-Nr-AL-Register: 02h)

Closes a connection to a TPM device with the specified parameters in the header. All data related to this connection to the device, such as allocated memory, are released. The registers in the configuration space of the TPM device are reinitialized to the reset status and the logical device is deactivated.

If the specified parameters in the header are not valid, or if closing of the connection to the TPM ends unsuccessfully, the function fails and returns corresponding *errorCode*.

BYTE MPCloseTPM (void);	
Input Parameters	*All necessary inputs are located in the driver header structure* (see Section 14.1.2).
Output Parameters	*None*
Return Value	*AL = return value of this function* One of the following values: **TPM_OK** **TPM_INVALID_ADR_REQUEST** **TPM_UNABLE_TO_CLOSE** **TPM_GENERAL_ERROR**

14.3.6.3 Function MPGetTPMStatusInfo (Function-Nr-AL-Register: 03h)

This function reads the current error and status information from the TPM device. All data related to this connection, such as allocated memory, are still valid.

If the specified parameters in the header are not valid or this device is not yet open, the function fails and returns an error flag.

DWORD MPGetTPMStatusInfo (void*)*;	
Input Parameters	*All necessary inputs are located in the driver header structure* (see Section 14.1.2).
Output Parameters	*None*
Return Value	*EAX = return value of this function* For the coding of the return value see Section 14.3.7.

14.3.6.4 Function MPTPMTransmit (Function-Nr-AL-Register: 04h)

Transmits the data from the input buffer (**pbInBuf*) to the TPM and reads the response from the TPM to the output buffer (**pbOutBuf*). After successful power-on and opening a TPM connection, the host can send the first request to the TPM by writing the bytes to the TPM.

This function is responsible for any blocking and deblock of the send and return parameters and error handling during the interaction with the TPM device over communication interface.

All vendor specific transport protocol information are added and removed by this function. The input and output buffer contains only TCG-Command-Param-Lists, these data streams are opaque to this function. This means that the TCG-Command-Param-Lists in these buffers will be not interpreted or reorganized by this function.

If no open connection to a TPM device is available, if it returns no response, if the function calling parameters are invalid, or the transmission of the data block to the TPM ends unsuccessfully, the function fails and returns corresponding *errorCode*.

BYTE MPTPMTransmit (**MPTPMTransmitEntryStruct** *lpTPMTransInfo*)*;	
Input Parameters	*ESI = pointer to a* TPMTransmitEntryStruct (see Section 14.3.5.5). *pbInBuf* Pointer to the input buffer containing the data (TCG command string) for the TPM device (see Section 14.3.5.1). *dwInLen* Length of the input buffer data (see Section 14.3.5.3).
Input/Output Parameters	*pdwOutLen* Pointer to store the length info of the received data (see Section 14.3.5.4). It also carries the size (input) of the OutBuf to store the response of the TPM device.
Output Parameters	*pbOutBuf* Pointer to the output buffer to store the data from the TPM device (see Section 14.3.5.2).
Return Value	*AL = return value of this function* One of the following values: **TPM_OK** **TPM_IS_LOCKED** **TPM_NO_RESPONSE** **TPM_INVALID_RESPONSE** **TPM_RESPONSE_TIMEOUT** **TPM_INVALID_ACCESS_REQUEST** **TPM_FIRMWARE_ERROR** **TPM_GENERAL_ERROR** **TPM_TRANSFER_ABORT**

14.3.7 Return Values for MPGetTPMStatusInfo (Function: 03h)

If the return value is "0", no error condition is active for this TPM connection. This status is the OK-Status of the TPM device.

Table 29: MPGetTPMStatusInfo Return Values (DWORD)

Bit	Descriptions
0	If set, a general error condition is active for this TPM connection. For details, evaluate the condition of the following error information (Bit 1:15).
1	Invalid status/error request access.
2	If set, a general firmware error occurred during start up of the TPM firmware.
3	Time out occurred during send process of the request sequence to the TPM device.
4	Response time out in TPM communication.
5	Transfer communication abort with the TPM device.
6	Reserved. This bit is read-only and has a value of 0.
7	Reserved. This bit is read-only and has a value of 0.
8	Reserved. This bit is read-only and has a value of 0.
9	Reserved. This bit is read-only and has a value of 0.
10	Reserved. This bit is read-only and has a value of 0.
12	Reserved. This bit is read-only and has a value of 0.
13	Reserved. This bit is read-only and has a value of 0.
14	Reserved. This bit is read-only and has a value of 0.
15	Reserved. This bit is read-only and has a value of 0.
16	If set, a general status information is available for this TPM. For details, evaluate the condition of the following status information (Bit 17:31).
17	The TPM device is not personalized (e. g., Endorsement Key pair is missing).
18	Integrity discrepancy in the TPM initialization.
19	Self-Test of TPM device complete.
20	Data transmission with TPM device active.
21	Reserved. This bit is read-only and has a value of 0.
22	Reserved. This bit is read-only and has a value of 0.
23	Reserved. This bit is read-only and has a value of 0.
24	Reserved. This bit is read-only and has a value of 0.
25	Reserved. This bit is read-only and has a value of 0.
26	Reserved. This bit is read-only and has a value of 0.
27	Reserved. This bit is read-only and has a value of 0.
28	Reserved. This bit is read-only and has a value of 0.
29	Reserved. This bit is read-only and has a value of 0.
30	Reserved. This bit is read-only and has a value of 0.
31	Reserved. This bit is read-only and has a value of 0.

15. Physical Presence

Start of informative comment

Physical presence is required in order to enable certain TPM commands. These commands are generally used to bypass Owner-authorized commands when the authorization data is unavailable or to set the TPM to an non-Owner state. Two methods are specified in this section. Others may be implemented if it can be demonstrated that the mechanism is tamper proof to the same extent as the integrity of the CRTM.

End of informative comment

The motherboard MAY provide a mechanism that provides proof of a human's physical presence to the Host Platform. There may be TCG specifications describing optional methods for managing an operating system to BIOS physical presence interface. The reader is directed to the TCG's set of documents for these specifications.

15.1 Physical Switch

A physical switch or jumper or momentary button that when activated provides a Physical Presence signal to the TPM. It MUST NOT be possible to generate this signal from software. This switch, jumper, or button MUST be in a location typically inaccessible to the user during the normal operation of the Host Platform; for example, a DIP switch connected to the motherboard which is within the Host Platform case.

15.2 Indication of Physical Presence from the CRTM

The CRTM MAY be designed to detect the user's physical presence and use the TSC_PhysicalPresence operation to indicate physical presence to the TPM. If a utility external to the CRTM is predicated upon an indication of physical presence, it MUST be designed such that it can only be executed if the user is physically present at the Host Platform (e.g., insertion of a floppy disk, a USB device, or pressing a button). The CRTM MUST perform one of the two following sequences based on the indication of physical presence:

1. Physical Presence is NOT indicated: Exit normally, processing the remaining portions of the Pre-Operating System State.

   In this option, prior to exiting, the CRTM MUST set the physicalPresenceMask flag appropriate to the design of the Host Platform. If physicalPresenceMask is TRUE, the CRTM MUST set the PhysicallyPresent to FALSE and PhysicalPresenceLock to TRUE.

2. Physical Presence IS indicated: Transfer control of the Host Platform to the utility that requires physical presence.

   Prior to transferring control of the Host Platform to the utility that requires physical presence, the CRTM MAY leave the PhysicalPresenceMask, PhysicallyPresent, and the PhysicalPresenceLock flags in any state appropriate for the design of the Host Platform and entry into the utility. However, upon exit from the utility, it MUST set the PhysicalPresenceMask flag appropriate to the design of the Host Platform. If PhysicalPresenceMask is TRUE, the CRTM MUST set the PhysicallyPresent to FALSE and PhysicalPresenceLock to TRUE.

15.3 Physical Presence via Operator Authorization

Start of informative comment

TPM Main Part 3 Commands, Section 5.7 TPM_SetOperatorAuth defines a method for establishing an operator authorization value.

End of informative comment

16. 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