TCG

TCG PC Client Specific Implementation Specification for Conventional BIOS

Specification Version 1.21 Errata
Revision 1.00
February 24th, 2012
For TPM Family 1.2; Level 2

Contact: admin@trustedcomputinggroup.org

TCG Published

Contents

Figures

Tables

Corrections and Comments

Please send comments and corrections 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 116 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 1.21; Revision 1.00.
3. The Platform Class for platforms adhering to this specification SHALL be registered with the TCG Administrator[1].
4. Host Platforms claiming adherence to this specification MUST be compliant with the TCG ACPI Specification, Version 1.00, Revision 1.00 or later.

[1] Platform classes are an enumeration managed by the TCG. The platform class value is normative to software, policy engines, software, etc. that need to define how to interpret information from platforms adhering to this specification. See section 11.3.4.1: Specification Event for more information.

1. Introduction and Concepts

Start of informative comment:

The Trusted Computing Group's architecture is platform-independent, intended to enhance trust in computing platforms. As such, the TPM 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 type of computing architecture. It is anticipated that companion implementation documents will be created for each architecture.

This document serves as implementation reference documentation for the 32-bit and 64-bit PC Client architectures. Specifically, this document defines:

1. Usage of PCR registers in the Pre-Boot state through the transition to Post-Boot state.
2. How the BIOS, or a component thereof, functions as the Static Core Root of Trust for Measurement (S-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 *TPM 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 TPM Main Specification will not be defined in this document. If there is a conflict in interpretation between this and the TPM Main Specification, the concept or functional description as defined in the TPM Main Specification takes precedence.

This specification also references other specifications as listed in Section 1.6 (External Specifications). The reader is expected to be familiar with the concepts and terminology contained in each 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 with a measurement log. While not trusted on their own, the measurement log contains a richer set of information than do 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 than 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 measurement 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 Client architecture are described in both Figure 1 and the descriptions. While the diagram in 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 platform 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.

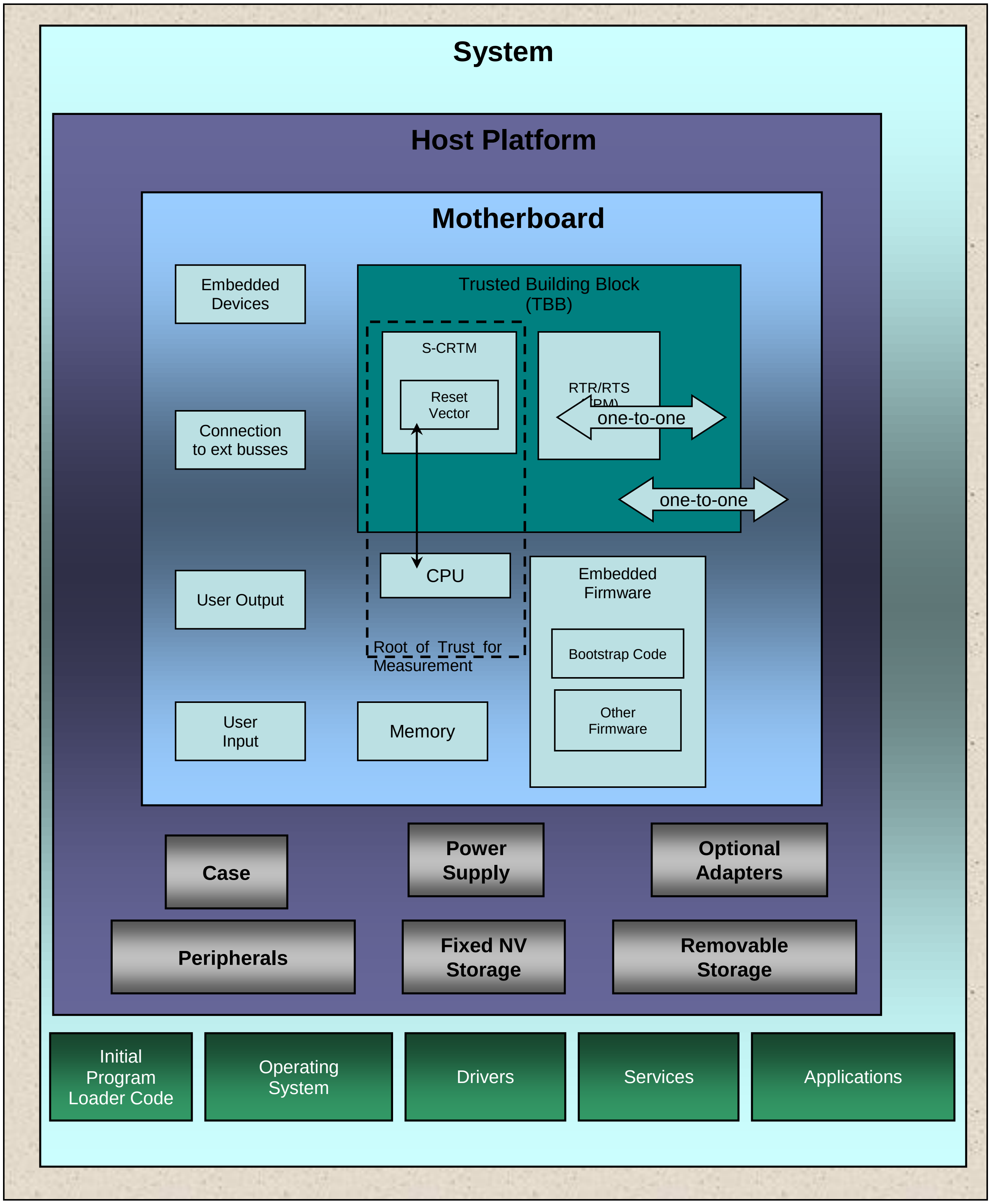

Figure 1: General architectural components of the PC Client

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 Main Specification identified in the TPM Dependency and Requirements section above.

1.2.2 Immutable

In this specification, *immutable* is defined that, in order to maintain trust in the Host Platform, only a Host Platform manufacturer-approved agent and method MUST perform the replacement or modification of code or data. This allows the manufacturer to control the upgrade method for the portion of the Host Platform that is the Static Core Root of Trust for Measurement (S-CRTM) with consideration for the security properties of the Platform's Protection Profile. See Section 3.3.1.2 (Static Core Root of Trust for Measurement (S-CRTM)).

1.2.3 Trusted Building Block (TBB)

The TBB consists of hardware and/or software that establishes trust (provides an integrity measurement) and provides connectivity between an S-CRTM, the TPM, the PC Motherboard, the platform reset, and the TPM physical presence signal.

Because the S-CRTM and the TPM are the only implicitly trusted components of the PC 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.

The TBB provides functionality that permits an entity to believe measurements that describe the current computing environment in the platform. An entity can assess those measurement results and compare them with values that are expected if the platform is operating as expected. If there is a match between the measurement results and the expected values, the entity can trust computations within the platform to execute as expected.

The S-CRTM initiates the measurement of the state of the hardware and software environment in a platform. Three data components are involved in an integrity metric. The first component is the method used to gather the data. The second component is predicted values of measured data in a platform. The third component is the actual values of measured data in a platform. Any integrity challenger needs to know about all of these components in order to make a decision about the integrity of the platform.

Measurements must be done in ways that ensure the validity of the collected data. Hence, the TBB provides a Root of Trust for the process of measuring integrity data. This is the purpose of the S-CRTM. The TBB's connections to the TPM, and PC Motherboard provide trusted pathways for information to flow between these components.

A PC Client platform has a TBB consisting of the S-CRTM and the TPM. A PC Client has only one TPM, one S-CRTM, and one TBB. The TBB has a one-to-one binding with the PC Motherboard.

Figure 1 depicts the "one-to-one" logical relationships between the TBB, the TPM, and the PC Motherboard. The components within the box labeled "TBB" are within the TBB—for example, the S-CRTM and the connections to the TPM and the platform are part of the TBB. Note that the removable storage, input devices, output devices, CPU, remaining portion of the BIOS, and supporting hardware are not part of the TBB. The supporting hardware includes components to connect memory and I/O controllers to the PC Motherboard.

1.2.4 Host Platform

The Host Platform is the entity that executes the Host OS, which executes, presents output from, and receives input for the Host Applications for the users (remote or local). The Host Platform includes the PC Motherboard, Host Platform's CPU, Host RTM, and Host TPM. The term "CPU" in this specification refers to the Host Platform's CPU unless otherwise stated.

1.2.5 Non-Host Platforms

This specification defines a Non-Host Platform as 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 Intelligent Platform Management Interface 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 true state of the platform. If a Platform Credential or Security Target is produced for this platform, they 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 compose the entire entity that performs actions for, or acts on behalf of, the user. The entity 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 and TPM Reset

Start of informative comment:

A Host Platform Reset is a hardware event that causes execution on the Host Platform’s CPU to permanently end its current instruction sequence and begin executing within the Core Root of Trust for Measurement (CRTM). This means that the CPU loses its entire context and begins execution at its reset vector. A Host Platform reset causes all Host Platform components to behave in their default, reset state.

Several events can cause Host Platform Reset, 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 OS to begin a new boot session; and initiated by the CPU during certain unrecoverable fault conditions. The Host Platform has a consistent behavior, from a trust perspective, regardless of the cause of reset performed.

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

Host Platform Reset only deals with establishing trust in the S-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.

The primary concern when 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.

TPM Reset is defined in the TPM Interface Specification, Section 1.1 (Terminology).

End of informative comment.

1.2.7.1 Types

Start of informative comment:

A Hardware Host Platform Reset occurs when a signal activates the reset signal of all Host Platform components. This may be a user-initiated or a software-initiated event triggered by a command to a hardware component that asserts the reset line. This includes resuming from S3.

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

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. For this specification, a reboot is equivalent to transitioning through a Soft Off state.

A TPM_INIT occurs on a Cold Boot, a Warm Boot, and upon resuming from S3 (sleep). The S-CRTM issues the TPM_Startup command, which tells the TPM whether to load saved state (for S3 resume) or not (for a boot).

Regardless of the types of Host Platform Reset described above, the normative text of this section applies to all of them.

End of informative comment.

1.2.7.2 Host Platform Reset

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

1.2.7.3 TPM Reset

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

1.2.8 PCI Option ROM Request for Reset

Start of informative comment:

The PCI Firmware Specification requires the actions in this section and, provided the BIOS performs this function, no further action is required. However, examples of non-compliant BIOS exist in the marketplace today. This section simply restates the requirement because it is important for security reasons.

End of informative comment.

If BIOS supports Expansion ROM Configuration Utility Code Management, as described in Section 5.2.1.24 of the PCI Firmware Specification, Version 3, upon return from the Expansion ROM configuration code, the BIOS MUST check the return status from the configuration utility and if the configuration utility requests a system reset, the BIOS MUST perform a Host Platform Reset.

1.2.9 Trusted Process

A Trusted Process is either a hardware-based or a software-based process within the Host Platform that is trusted without the need for performing further inspection as expressed by the Host Platform Certificate. The Root of Trust for Measurement (RTM) is an example of a Trusted Process within its trust domain. (e.g., Static RTM or Dynamic RTM).

1.2.10 TPM Enumeration

Start of informative comment:

Generally, OS loader components use the Pre-Boot INT 1A TCG functions (described in Section 13 (Application Level Interface – INT 1A TCG Functions)) to identify and interact with the TPM, and a fully loaded OS uses the ACPI table entry (described in Section 10.1 (Device Object for TPM)) to identify the TPM and an OS resident TPM driver to interact with the TPM.

Platforms with a TPM intending for a fully loaded operating system to use the TPM, enumerate the TPM device in the ACPI tables of devices for the OS.

Because some platform manufacturers may ship a platform with an operating system that does not have an OS TPM driver, the platform manufacturer may provide a mechanism to avoid enumerating the existence of the TPM on the platform so users do not see the TPM device without an OS driver in an operating system device list.

Mechanisms to control enumeration of the TPM to an operating system are platform manufacturer-specific. An example is a BIOS configuration option to enumerate or not enumerate the TPM to the OS.

End of informative comment.

A TPM is *enumerated* if the platform firmware indicates the presence of the TPM device to the OS. A TPM is not *enumerated* if the platform firmware does not indicate the presence of the TPM device to the OS. See Section 9 (TPM Enumeration) for more information.

1.2.11 TPM Discoverability

Start of informative comment:

Because some platform manufacturers may provide a mechanism to not enumerate a TPM, an alternative mechanism is used to indicate if a TPM is physically present on a platform and the Host Platform may be configured through some mechanism provided by the platform manufacturer to use the TPM.

As part of inventorying their computer system capabilities, some information technology staff in an enterprise may want to discover which hardware platforms have a TPM, whether the TPM is currently enumerated to the operating system or not. As of version 1.21 of this specification, the existence of the TCPA ACPI table means a mechanism exists on the platform for a platform owner to make the TPM enumerated to the operating system and available for use.

End of informative comment.

A TPM *is discoverable* on a Host Platform if it is physically present and if the TPM could be enumerated by defined platform owner action, regardless of whether the TPM is currently enumerated. A TPM is *not discoverable* on a Host Platform if it is not physically present on the platform or if the TPM is physically present, but system components prevent the platform owner from taking any platform manufacturer-defined action to enumerate the TPM.

1.2.12 Roots of Trust for Measurement

The terms Root of Trust for Measurement and CRTM within this specification refer to those entities that are associated with the Host Platform.

1.2.12.1 Root of Trust for Measurement (RTM)

The RTM is implicitly trusted. Trust in this component may be expressed in the Host Platform Certificate. The RTM 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.

1.2.12.2 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.

1.2.12.3 Privacy Setting and the Scope of the RTM

Start of informative comment:

If the Host Platform implements privacy settings using the command method (as defined in the TCG Physical Presence Interface Specification) for the indication of physical presence, those settings are under the control of a process within the S-RTM and are measured as part of the S-RTM. This is to provide verifiers (including the user or operator) with a method to validate that their privacy settings are respected and enforced. For example, if the platform uses the BIOS to detect the Operator by someone pressing a "function" key while the platform is under control of the BIOS, that detection code and the code that sends the physical presence commands must be measured unconditionally. Unconditionally measuring the physical presence command code on every boot allows a verifier to validate the code without needing to perform a physical presence command to obtain the code measurement. The code is not measured with a special event; it is measured as part of other S-RTM measurements using events like the S-CRTM version identifier (EV_S_CRTM_VERSION) or a measurement like EV_POST_CODE.

End of informative comment.

On platforms that use the command method for asserting physical presence to the TPM, the platform MUST measure the components that perform this function on every boot as part of the S-RTM using one or more of these events:

1. As part of the S-CRTM's version identifier measurement, using the event type EV_S_CRTM_VERSION in PCR[0] as described in Section 3.3.3.1 (PCR[0] – S-CRTM, POST BIOS, and Embedded Option ROMs).

   Note: This does not actually measure the physical presence command code because it is part of the implicitly trusted S-CRTM which may be measured as a version identifier as described in Section 3.3.3.1 (PCR[0] – S-CRTM, POST BIOS, and Embedded Option ROMs).
2. As part of the S-CRTM using the event type EV_S_CRTM_CONTENTS in PCR[0] as described in Section 3.3.3.1 (PCR[0] – S-CRTM, POST BIOS, and Embedded Option ROMs).
3. As part of the BIOS using the event type EV_POST_CODE in PCR[0] as described in Section 3.3.3.1 (PCR[0] – S-CRTM, POST BIOS, and Embedded Option ROMs).

1.2.13 Boot State Transition

The transition between Pre-Boot and Post-Boot states is the first invocation of INT 19h or equivalent.

1.2.14 Establishing the Chain of Trust

1.2.14.1 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 Version 1.2, Part 1, Design Principles, Section 11.2 (RTR to Platform Binding).

1.2.14.2 Binding Methods

Start of informative comment:

The method of binding the TPM to the PC Motherboard is an architectural and design decision made by the platform manufacturer and is not specified here. The two types of binding are: Physical and Logical. Physical binding relies on hardware techniques, while Logical binding relies on cryptographic techniques. The requirements for the strength of each binding are within the scope of a 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.

End of informative comment.

1.2.15 Locality States

Start of informative comment:

A general description of locality can be found in the TPM Main Specification, Part 1, Design Principles, Section 16 (Proof of Locality) and in the TPM Interface Specification, Section 9 (Locality). However, a brief overview of how it is used is provided here as well. In a platform with multiple trusted processes, locality provides an indication of the source of the command. It also provides a mechanism for sharing the TPM's resources across and between multiple security domains.

Indication of the Command's Source

The TPM is accessed using a set of address registers. Each set of registers creates a group with the registers within each group being mirrored (generally, but there are minor exceptions). Because the registers are mirrored, software (the driver) accesses the TPM the same way regardless of which group it uses. Groups are defined by an address range. Each address range corresponds to a locality. Note that the TPM provides no access control over localities. This is the function of the surrounding hardware or trusted processes. The TPM simply assumes that if it receives a request at any locality, the process sending the request is authorized to do so at that locality. Some commands are restricted to some localities and most objects can have locality restrictions placed onto them. For example, it is possible to create a key that is only usable at Locality 0 or a Sealed blob that can only be unsealed from Locality 2.

Sharing of the TPM

To access the TPM, a process (i.e., some entity operating at a specific locality) must request use of the TPM at that process's locality. The process then uses the TPM for one or more operations (essentially a session). The software should then release the TPM. This allows the TPM to be shared between the various processes within the platform. To accomplish this "arbitration" scheme, the TPM Interface Specification specification defines the concept of locality. When a process wants to use the TPM, it requests use at a specific locality. When the process has completed its use of the TPM, it releases the locality. It is not until the TPM "grants" use of the TPM from a free state to the requesting locality that it may send commands to the TPM.

Legacy Locality

The Family 1.1 TPM Main specifications did not define a TPM interface, rather leaving that to the various TPM manufacturers. Those manufacturers chose to access the TPM using I/O ports. Because of the need to allow software written to the TPM 1.1's interface to function on some Family 1.2 TPM devices, the TIS specification defines a port I/O interface. However, there is no arbitration scheme available for the Family 1.1 TPM. For this reason, it was decided that only while the TPM is at no active locality (i.e., not being used by a process) can it be accessed using the legacy port I/O interface.

General Description of the CRTMs

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: Relationship between Static and Dynamic RTMs, these two RTMs and their respective chains of trust have no relationship to each other except that they are both rooted at the same Root of Trust for Storage (RTS) and Root of Trust for Reporting (RTR) (i.e., the TPM). This is an important consideration in that the trust of each is dependent only on the trust in the common RTS/RTR and their own RTM.

End of informative comment.

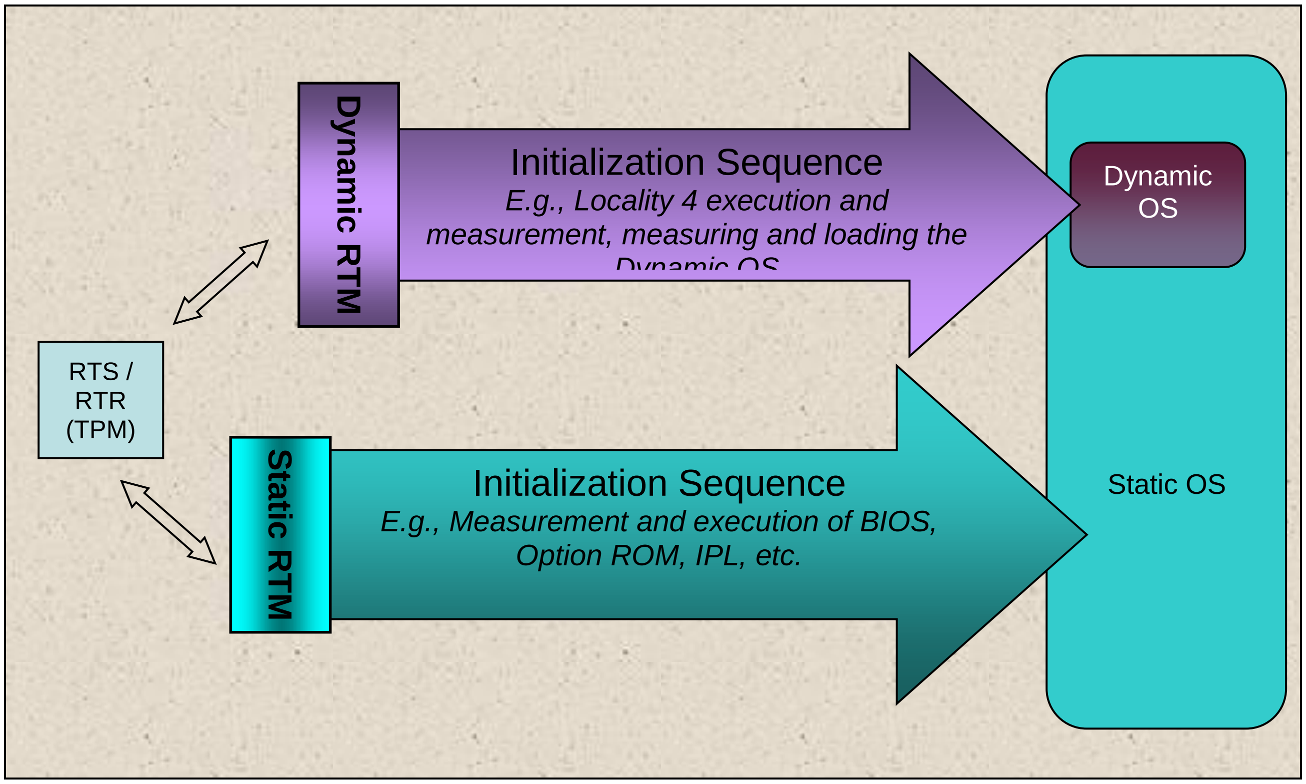
Dynamic RTM
Initialization Sequence
E.g., Locality 4 execution and measurement, measuring and loading the Dynamic OS
Dynamic OS
RTS / RTR (TPM)
Static RTM
Initialization Sequence
E.g., Measurement and execution of BIOS, Option ROM, IPL, etc.
Static OS

Figure 2: Relationship between Static and Dynamic RTMs

1.2.15.1 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 while verifiable within its own domain, is dependent on assumptions about its environment such as whether 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, S-RTM or some subset of both.

An 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 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 chain of trust) before proceeding to verify the security assertions of the D-RTM.

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 the union of their trust statements within a larger environment.

End of informative comment.

1. Architecturally, 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 implementations MAY create a dependency between them. This dependency, if any, SHOULD be represented in the Host Platform Certificate.

1.2.16 System and TPM Power States

Start of informative comment:

For PCs, a power management standard called ACPI defines a set of power states that each have different performance and power requirements. PC Client platforms that comply with this specification will support some or all of these power states. In general, the ACPI specification describes three types of power states: global states, system states, and device states. Refer to the ACPI specification for a full description.

System States

System states describe the power state of the entire system. Short descriptions of system states are:

Legacy – Intended for use by non-ACPI operating systems

G0 (or S0) – Working state (faster response times, high power usage)

G1 – Sleep states, which include several different system states:

S1 – Stand-by with low wakeup latency sleeping state

S2 – Stand-by with CPU context lost sleeping state

S3 – Suspend to RAM (system memory is preserved, other state is lost)

S4 OS Initiated – The OS suspends system state to a persistent storage and restores it later

S4 BIOS Initiated – The BIOS suspends system state to persistent storage and restores it later

G2 (or S5) – Soft Off (the system is restarted to return to the working state)

G3 – Mechanical Off (the system is restarted to return to the working state)

G0 and S0 are different names for the same state. Likewise, G2 and S5 are different names for the same state.

This specification uses the term S0 to mean both S0 and G0. Because G2, G3, S4, and S5 all refer to states where the system is not running, the term “Off” is used for all of them in this specification.

Device States

Device states describe the power use and context maintained for device on a system. Short descriptions of device states are:

D0 – Generally fully on (full functionality, probably full power use)

D1 – A lower power state with potentially longer latency

D2 – An even lower power state with potentially longer latency

D3 – Generally off (device context is lost)

The only transitions allowed for system states and devices states are those defined in the ACPI specification. There is not a direct correspondence between device states and system states; for example, a device may be state D3 while the system is in state S0.

TPM Device State

The TPM device per the TPM Interface Specification, Section 4 (Power Management) behaves identically in states D0 (fully-on), D1 (device-specific low power state), and D2 (another device-specific lower power state). Note: Implementing D1 and D2 for a TPM is not recommended. The TPM device is not allowed to exit state D3 without receiving a TPM_INIT.

The TPM has commands designed to deal with saving TPM state before placing the TPM in the D3 state and restoring TPM state when leaving the D3 state. This specification has the OS issue a TPM_SaveState command before placing the TPM in D3. After placing the TPM in D3, the system may enter the S3 state. Upon resuming from S3, the TPM is initialized and started so the saved state is restored. Other scenarios are possible when placing the TPM in D3, but they may result in the TPM not being usable later without a transition to S5 and are not discussed in this specification.

End of informative comment.

1.2.17 General Host Platform Power Requirements

Start of informative comment:

There is no required behavior during any power state as long as 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-S2. However, for example, the TPM may be implemented such that auxiliary power is required to maintain saved state (like PCR registers) during the system state S3 or the TPM device state D3. In this case, a Host Platform incorporating this type of TPM needs to provide necessary power to the TPM during D3 and S3, while Host Platforms incorporating TPMs using flash memory or other non-volatile (NV) storage technology will not require power during D3 or 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-S2, of course). Some TPM manufacturers may choose to provide this features as a “value add.” Those TPMs may require auxiliary power during non-S0 power states.

This specification does not define a mechanism to power down a TPM and restore its saved state while the system is in the ACPI Legacy state. The only mechanism defined in this specification to restore a TPM saved state is to transition out of the ACPI S3 state to S0. While in ACPI Legacy mode, the system should keep the TPM powered so its state is preserved.

End of informative comment.

1.3 Overview of Measurement Process

Start of informative comment:

This section uses, but does not define, concepts that are explained later in this specification.

This specification defines measurement as the process of hashing various components of the boot process, extending the results in the TPM and logging the component's measurement in the measurement log in the Host Platform memory. The specific components, order of measurements, and storage locations are specified in the normative text in subsequent sections.

The general sequence of operations is diagramed below. While the sequencing in this diagram is described in the normative within this specification, the diagram is informative only. This general description makes no attempt to itemize the optional and mandatory components or sequences.

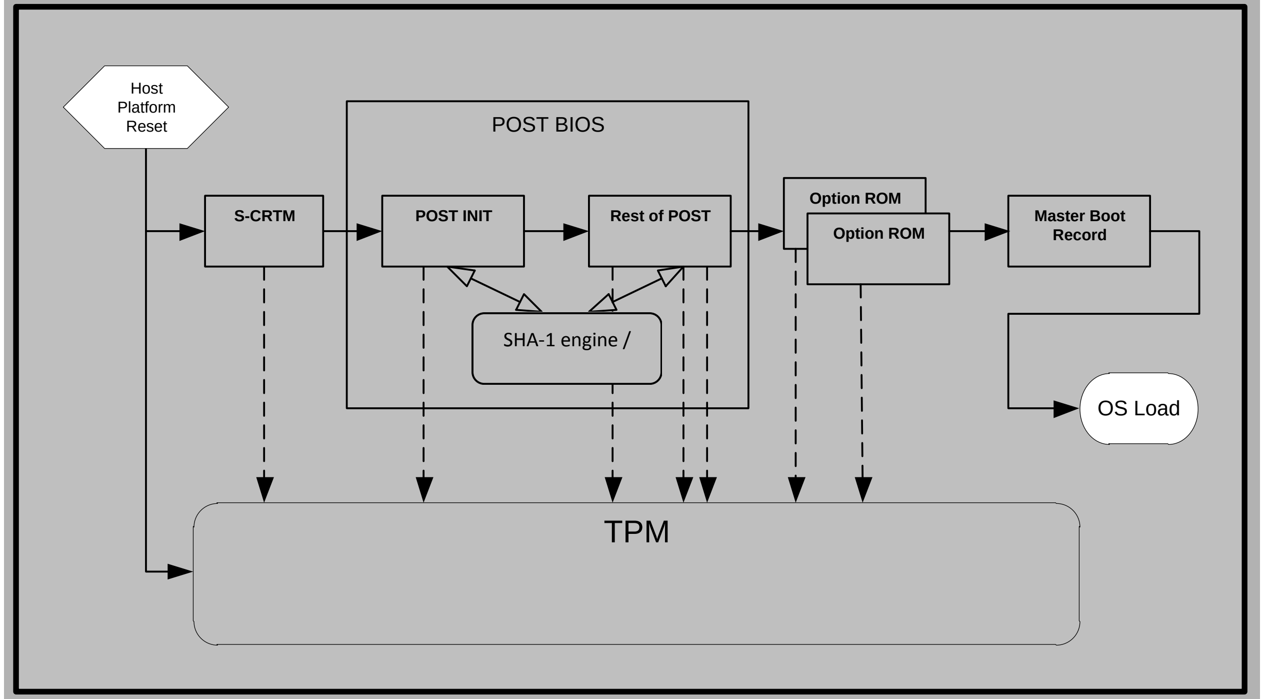

Figure 3: Example boot flow

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 is a set of hash functions to enable measurement:

TPM_SHA1Start; TPM_SHA1Update; TPM_SHA1Complete; and TPM_SHA1CompleteExtend

These functions are provided for environments where 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 configured. In this implementation of this environment, it would be faster to send the data to be measured to the TPM rather than have even a fast Host Platform CPU perform the hash using only registers.

It must be kept in mind that the TPM interface and the TPM are comparatively slow, and judicious use of the TPM-based hashing functions is prudent. It is best to utilize these commands for as few measurements as possible and only until memory is enabled, then the BIOS should switch over to using its own SHA-1 engine as quickly as possible, thereby improving boot-time performance.

The goal of utilizing the TPM's hashing function 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 limited 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_SHA1Start, TPM_SHA1Update, TPM_SHA1Complete, and TPM_SHA1CompleteExtend (hereafter referred to as TPM_SHA1*) commands. While the use of the TPM's hashing capability improves performance during the pre-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: Example boot flow, 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 code's security components, the S-CRTM would measure only the portions of POST that are called "POST Init" and "SHA-1 engine" in the diagram. The only requirement of the POST Init module is that it measures any component prior to transferring control 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 and SHA-1 engine were measured prior to enabling 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:

The definitions below 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 (BIOS Recovery Mode).

End of informative comment.

1.4.2 BIOS Resume

Start of informative comment:

BIOS resume is a resume from S2 or S3.

End of informative comment.

1.4.3 Dynamic OS

Start of informative comment:

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

End of informative comment.

1.4.4 Dynamic Root of Trust for Measurement (D-RTM)

Start of informative comment:

The D-RTM is the chain of measurements of platform state that begin after the execution of TPM_HASH_START. The measurement chain continues with components measuring subsequent components and configuration data before passing control to them. D-RTM measurements are recorded in PCRs[17-22]. The resulting OS that is loaded and may be measured in the D-RTM PCRs is called the Dynamic OS. (Mandatory measurements of the Dynamic OS loader are out scope for this specification.)

End of informative comment.

1.4.5 Host Platform CPU

Start of informative comment:

The Host Platform CPU is defined as 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.

1.4.6 Initial Program Loader Code (IPL Code)

Start of informative comment:

The IPL code is the area of the IPL Image containing 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.

1.4.7 Initial Program Loader Data (IPL Data)

Start of informative comment:

The IPL Data is the area of the IPL Image containing only data. For example, this area contains the Master Boot Record’s (MBR) partition table.

End of informative comment.

1.4.8 Initial Program Loader Image (IPL Image)

Start of informative comment:

The IPL Image is the area containing the IPL Code and any IPL Data. An example of an IPL Image is the first section of a hard disk’s MBR. This area contains both the executable IPL Code and the partition table.

End of informative comment.

1.4.9 Manufacturer Approved Environment (MAE)

Start of informative comment:

A MAE is an environment that 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.

1.4.10 Measure

Start of informative comment:

The term “measure” means the act of calculating a hash (SHA-1) digest of some code or data, the subsequent extension of the digest in an appropriate PCR within the TPM, and the addition of a log entry about the data and its digest in the measurement log using the appropriate event log structure.

End of informative comment.

1.4.11 Measurement

Start of informative comment:

The measurement of a component is the SHA-1 hash of the component. The measurement of a chain of components is the resulting PCR value of extending each component measurement successively in a PCR. Sometimes a measurement refers to the resulting set of PCR values after successively extending components into a set of PCRs.

End of informative comment.

1.4.12 Non-Manufacturer Approved Environment (NMAE)

Start of informative comment:

An NMAE 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 an NMAE.

End of informative comment.

1.4.13 PC Motherboard

Start of informative comment:

The PC Motherboard is the object that is supplied by the manufacturer that is composed of the TBB and other components physically or logically attached and supplied by the manufacturer.

End of informative comment.

1.4.14 Platform Manufacturer

Start of informative comment:

The Platform Manufacturer is the entity responsible for completing the manufacture of the final system prior to delivery to the end customer. The platform manufacturer is responsible for ensuring the platform and all its components comply with all requirements in this specification. Modern systems often involve a manufacturing supply chain and the platform manufacturer may engage with suppliers to implement earlier steps in the manufacturing process. As the owner of the manufacturing process, the platform manufacturer attests to the validity of the system and its components.

End of informative comment.

1.4.15 Post-OS State

Start of informative comment:

The Post-OS state is the state of the system after the invocation of the first INT 19h or its equivalent. This may include OS, PARTIES, diagnostics, etc.

End of informative comment.

1.4.16 Power On Self-Test (POST)

Start of informative comment:

The Power On Self-Test (POST) is a generic term for BIOS code that runs after a Host Platform Reset.

End of informative comment.

1.4.17 Pre-OS State

Start of informative comment:

The Pre-OS state is the state of the system prior to the invocation of the INT 19h or its equivalent.

End of informative comment.

1.4.18 Static Core Root of Trust for Measurement (S-CRTM)

Start of informative comment:

The S-CRTM is the executable component of the RTM that gains control of the Host Platform upon a Host Platform Reset. See a more detailed description in Section 3.3.1.1 (Initial TBB Control and Host Platform Reset).

End of informative comment.

1.4.19 Static OS

Start of informative comment:

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

End of informative comment.

1.4.20 Static Root of Trust for Measurement (S-RTM)

Start of informative comment:

The S-RTM is the chain of measurements of platform state that begin at Host Platform Reset (e.g., power-on or system restart) with the exception of a Host Platform Reset during S3 resume. The measurement chain continues with components measuring subsequent components and configuration data before passing control to them. S-RTM measurements are recorded in PCRs[0-15]. The resulting OS that is loaded and measured in the S-RTM PCRs is called the Static OS. (Mandated measurements in this specification include the Static OS loader, but how much of the OS is measured in PCRs is OS implementation-specific.)

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, refer to the informative and normative comments contained in this document. For example, "TCG PC Client Specific Implementation Specification for Conventional BIOS TCG, Version 1.21 as released."

This specification defines only functional aspects of the concepts and implementation of a TPM, S-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)

Start of informative comment:

The TPM Interface Specification (TIS), Version 1.21, Revision 1.00 is the "companion" specification to this specification. The TIS defines the programmatic interface to the TPM and the method by which it is connected (i.e., which local Host Platform bus) to the Host Platform.

End of informative comment.

1.5.3 TPM Main Specification

Start of informative comment:

Refers to the TCG TPM Main Specification, Version 1.2 as released. The specification has three parts. Unless a specific part is referenced, this reference refers to all parts of the specification.

End of informative comment.

1.5.4 TCG TSS Specification

Start of informative comment:

Refers to the TCG TPM Software Stack (TSS) Specification, Version 1.2 as released.

End of informative comment.

1.5.5 TCG ACPI Specification

Start of informative comment:

Refers to the TCG ACPI Specification, Version 1.0, Revision 1.00, August 8, 2005 as released. The specification is a server work group specification and is listed as "Server Work Group ACPI General Specification, Version 1.0."

End of informative comment.

1.5.6 TCG Physical Presence Interface Specification

Start of informative comment:

Refers to the TCG Physical Presence Interface Specification, Version 1.20, Revision 1.00 as released.

End of informative comment.

1.5.7 TCG Platform Reset Attack Mitigation Specification

Start of informative comment:

Refers to the TCG Platform Reset Attack Mitigation Specification, Version 1.00, Revision 1.00 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 (ACPI) Specification

Revision 2.0, July 27, 2000 or later.

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 (BIS) Application Programming Interface

Version 1.0. (The specification defines the BIS certificate.)

1.6.5 System Management BIOS Reference Specification

Specification number dsp0134

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

1.6.9 PCI Firmware Specification

Version 3

1.6.10 Extended System Configuration Data Specification (ESCD)

Version 1.02A

Referred to as "ESCD" in this specification.

1.6.11 Trusted Configuration Space for PCI Express ECN

March 23, 2005; Updated July 1, 2005

2. Host Platform Setup and Configuration

Start of informative comment:

This section provides guidance to platform manufacturers regarding how to handle measurements for setup utilities.

End of informative comment.

2.1 Pre-Boot ROM-Based Setup

Entry into any PC Motherboard or Option ROM Setup Utility MUST be measured unless the Host Platform unconditionally performs a platform reset upon completion of any PC Motherboard or Option ROM setup. If measured, entry into a ROM-Based Setup Utility MUST be measured as EV_ACTION event “Entering ROM Based Setup” in PCR[1]. See Section 3.3.3.2 (PCR[1] – Host Platform Configuration). If measured, entry into an Option ROM-Based Setup Utility MUST be measured as EV_ACTION event “Entering ROM Based Setup” in PCR[3]. See Section 3.3.3.4 (PCR[3] – Option ROM Configuration and Data).

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 PC Motherboard-based BIOS or Option ROMs.

2.2 Post-Boot ROM-based Setup and Other Post-Boot Setup Utilities

Start of informative comment:

An example of Post-Boot ROM-based setup utility is a platform manufacturer-provided setup tool accessed via pressing a keyboard hot key during boot. An example of other Post-Boot setup tools is code read from a source other than platform manufacturer-provided ROM, like a partition of a hard drive or a PARTIES partition, which performs setup or configuration actions.

End of informative comment.

A Post-Boot ROM-Based Setup Utility is code stored in the platform ROM that provides setup or configuration functionality after the boot process exits the Pre-Boot environment.

Other Post-Boot Setup Utilities are any code from a source other than the platform ROM that provides setup or configuration functionality after the boot process exits the pre-OS environment.

Before the setup utility is called, it MUST be measured in PCR[4] as IPL Code using the event type EV_IPL per Section 3.3.3.5 (PCR[4] – Initial Program Loader (IPL) Code).

3. Roots of Trust Requirements

3.1 Summary

Start of informative comment:

Traditional host software (including host software that is expected to utilize the features provided by this specification) may have only one Root of Trust and one linear transitive trust chain starting from that root. In this model, the lowest layer in the software stack (i.e., the OS kernel or a Hypervisor) is the end-point of the single transitive trust chain. Here this lowest layer (called the kernel for convenience) is expected to be the only entity on the platform that communicates to the TPM and is expected (using methods such as rings, paging, etc.) to protect access to the TPM. Therefore, in this model there is no need to differentiate between the identity (or originator) of a TPM command since it can always be assumed the source of the TPM commands is the one kernel operating in the platform. This is architectural model of TPM Version 1.1.

TPM Version 1.2 introduces an architectural model where there may be two Roots of Trust, each with its own independent transitive trust chain. The first Root of Trust is the one mentioned above. It is called the "Static Root of Trust" because the transitive trust chain starting from this root starts a platform reset and is persistent (or static) until the next platform reset (or power down). The second Root of Trust begins at an arbitrary time chosen by either platform components or other software on the platform (e.g., the kernel) by executing a CPU instruction. This instruction is the start of the second Root of Trust and is called the Dynamic Root of Trust. Because this new Root of Trust starts with an instruction (rather than the platform reset), this Root of Trust can be re-initiated any number of times during one instance of the static Root of Trust. This is why this Root of Trust is called the Dynamic Root of Trust.

With two independent Roots of Trust (and the resulting two independent transitive trust chains that can be called environments) with a single TPM on the platform, the TPM needs a method to distinguish the source of a command so that TPM resources such as keys can be restricted and isolated to each environment. Because the source of the command is the command's locality, the term *locality* is used to distinguish between these different environments.

While the general model of the Static Root of Trust environment is a simple chain starting from the Static Root of Trust for Measurement, the Dynamic Root of Trust environment may be more complex. For example, a virtualized environment may allow sharing the TPM's resources. In addition, as it launches independent of the platform reset and possibly after many other processes are executing, its launch may require multiple steps. For these reasons, the Dynamic Root of Trust environment is allocated multiple localities.

Locality is implemented using address ranges. The various platform components or kernel(s) can then restrict access to localities from unauthorized platform entities using simple address ranges or paging. Each locality is assigned a single 4K page. The TPM Interface Specification, Section 9.1 (TPM Locality Levels) provides the normative reference to the assigned addresses. They are shown here in Table 1 for quick reference.

Resources such as keys, NV areas, PCR reset, and extend capabilities can be associated with one or more localities. While the exact use is beyond the scope of this document, Table 1 highlights some expected uses. It is important to note that the assignment of the reset capabilities for PCRs is a critical component for establishing the Dynamic Root of Trust.

The TPM itself enforces no access control between the execution environment and locality – each locality's access control is outside the TPM. Locality is enforced by the platform's chipset that connects the CPU to the TPM. The TPM simply assumes that the appearance of any command at any locality is from the associate environment.

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.

Locality	Address range	Entity in control	PCRs that can be reset by the locality	PCRs that can be extended by this locality
0	FED4_0xxxh	Static CRTM Static OS	16, 23	0 – 16, 23
1	FED4_1xxxh	Dynamic OS	16, 23	0 – 16, 20, 23
2	FED4_2xxxh	Dynamic OS	16, 20 – 23	0 – 23
3	FED4_3xxxh	An auxiliary level of a trusted component	16, 23	0 – 20, 23
4	FED4_4xxxh	Dynamic CRTM	16 - 20, 23	0 –18, 23

Table 1: Summary of locality usage

End of informative comment.

3.2 Locality State Requirements

3.2.1 For Pre-OS Environment

Start of informative comment:

Before sending a command to the TPM, the software or other platform components must set the TPM to operate at the desired locality. The TPM can be set to only one locality at a time. The TPM also has a state where there is no active locality set, called Locality None. The TPM can transition the active locality back and forth between different localities or Locality None. See the TPM Interface Specification for details about how these transitions are performed.

This section specifies the TPM's locality for pre-OS environment as it transitions to the Static OS.

It is important to note that prior to TPM version 1.2, the TPM typically used non-standardized I/O ports rather than memory-mapped ranges. To accommodate this, the TPM supports the no locality state. Commands sent using these legacy I/O ports must be done while the TPM is in the no locality state. Some operating systems and drivers will continue to support and require access to the TPM using the legacy I/O port and therefore will require the legacy locality. These drivers will not be designed to take or release the TPM's locality. These drivers will hang if the TPM is set at Locality 0 when these drivers attempt to access the TPM. For this reason, this specification allows the selection of the Legacy locality environment as depicted in Figure 4.

This section contains two normative diagrams:

Figure 4 shows the requirements for Host Platforms designed for v1.1b TPM compatibility. This implementation is for platforms whose components (including OS loaders) were designed prior to TPM 1.2's introduction of more than one environment. This implementation is deprecated.

Figure 5 shows the requirements for Host Platforms whose components are all compatible with v1.2 TPMs. These platforms' components (including OS loaders) can accommodate the Dynamic Root of Trust environment because the components know how to manage localities. Figure 5 depicts the environment where the TPM supports only the 1.2 memory-mapped I/O interface. Because components do not use legacy mode to communicate with the TPM, there is no reason to leave Locality 0 during the boot process. The static OS should set the TPM to no active locality state when it is not in use by the Static OS.

Note: Per section 15.1.4, the BIOS uses Locality 0 to access the TPM.

End of informative comment.

1. The platform MUST support the environment depicted in Figure 5.
2. The Platform MAY support the environment depicted in Figure 4.
3. If the platform supports the environment depicted in Figure 4, it MUST provide some means (e.g., a BIOS setup setting, PC Motherboard jumper, etc.) of selecting the environment.
4. In all environments
   a. the BIOS MUST transfer control to Option ROMs with the TPM in Locality 0, and
   b. the Option ROM MUST transfer control back to the BIOS with the TPM in Locality 0.

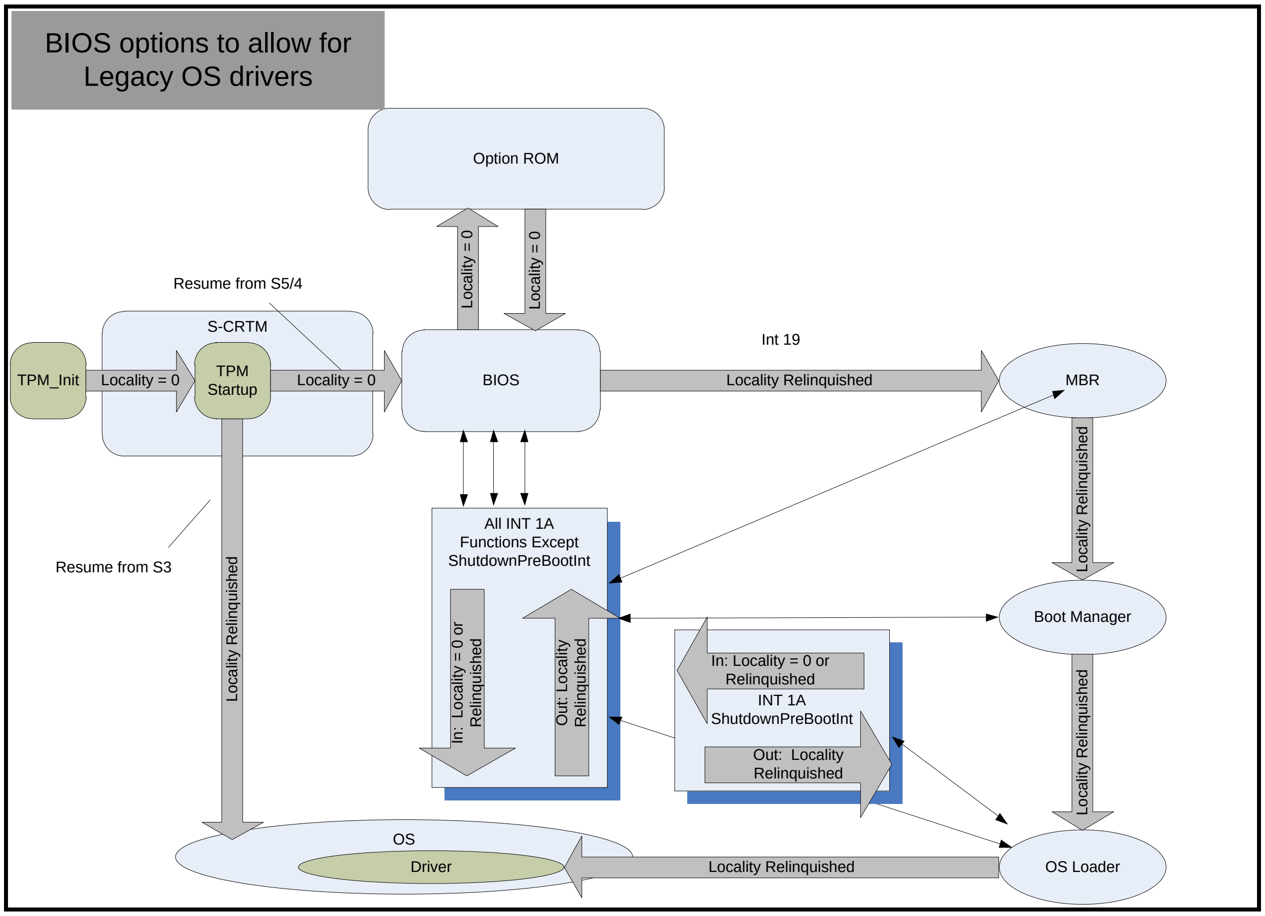

Figure 4: Legacy locality

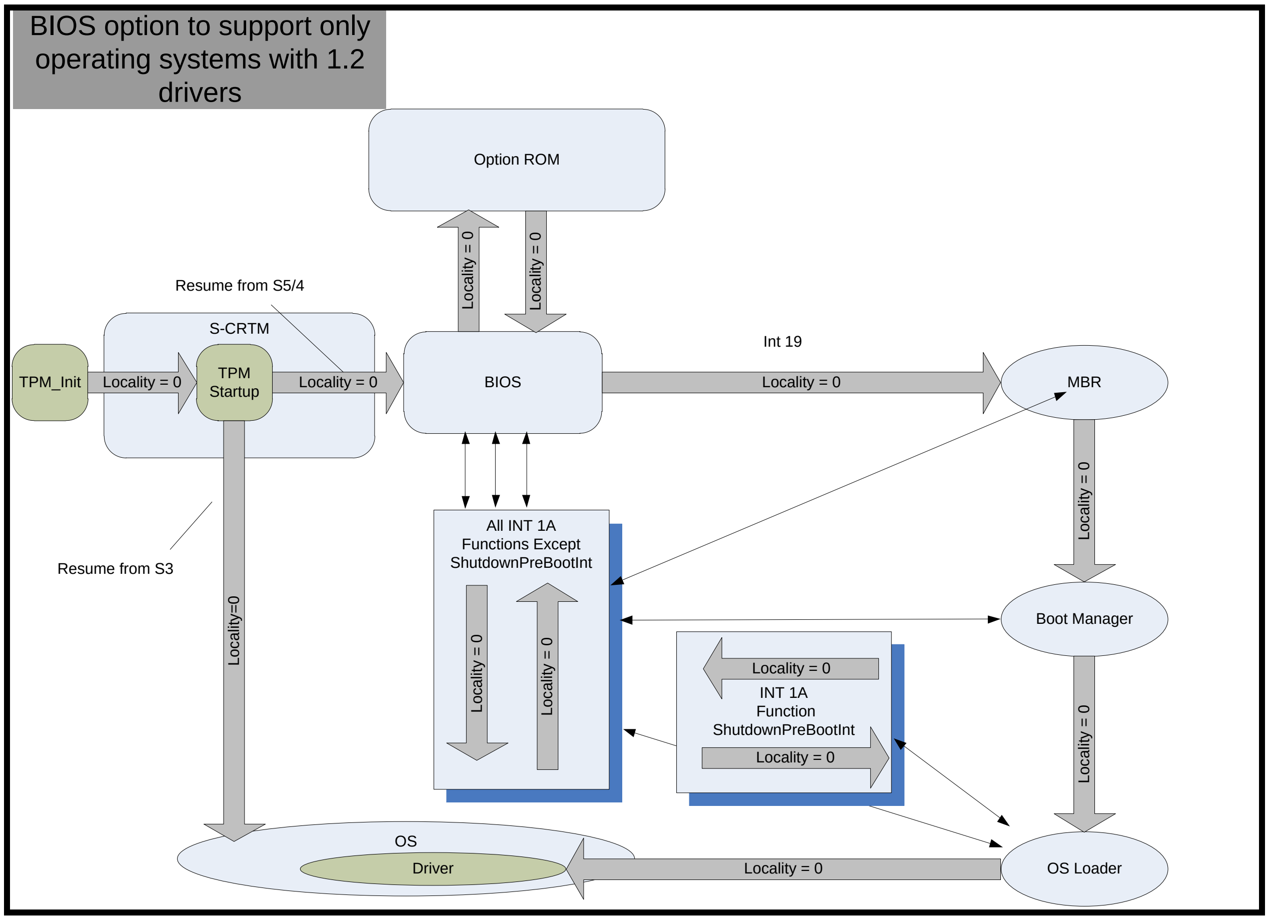

Figure 5: TPM 1.2 interface only

3.2.2 For the OS Environment

Start of informative comment:

Because a Static Root of Trust environment may be hosted in a virtualized environment, the Static OS should not assume sole access to the TPM. This is one of many reasons why the TPM is allowed to switch localities. All environments that use the TPM should request use of the TPM; use it when granted; then release it when done using the TPM. This is similar to any non-preemptive environment. The request, granting and release of the TPM's locality is done at the TPM's hardware interface layer and is expected to be very fast so there is little, if any, perceivable latency. Further, it is not expected that software will constantly use the TPM; rather, the use of the TPM will be sporadic during normal runtime operations.

End of informative comment.

1. Before sending a TPM command, an entity using the TPM MUST first verify that the TPM is in a locality the entity is authorized to access.
2. If the TPM is not in the correct locality, the entity using the TPM MUST request use of the TPM's locality or seize the locality before it may send a command. When granted, the entity is allowed to send commands to the TPM at the granted locality.
3. When an entity is done using the TPM, it SHOULD release the locality.

3.3 Locality for the S-RTM Environment

3.3.1 Concepts

Start of informative comment:

The S-RTM environment provides the Root of Trust for the components of the Host Platform as they are initialized following a Host Platform Reset. The goal of the S-RTM is to provide an unbroken chain of measurements for components that executed on the platform or security-relevant configuration data that may have influenced platform state. By assessing the chain of components, an entity may establish trust in the current state of the platform. The S-RTM is the default RTM for all components because the S-RTM measurements occur during boot by default. The S-RTM uses Locality 0 and the memory addresses associated with Locality zero, or for compatibility with TPM versions prior to 1.2, it may also use legacy port-based I/O when no active locality is set to record measurements in the TPM or send other TPM commands. See the TPM Interface specification for definitions of locality and memory addresses.

End of informative comment.

3.3.1.1 Initial TBB Control and Host Platform Reset

Host Platform Reset MUST cause the Host Platform to begin execution within the S-CRTM.

3.3.1.2 Static Core Root of Trust for Measurement (S-CRTM)

The Static Core Root of Trust for Measurement (S-CRTM) MUST be an immutable portion of the Host Platform's initialization code. See Section 1.2.2 (Immutable).

Note The trust in the S-RTM environment is based on the S-CRTM. The trust in all S-RTM measurements is based on the integrity of the S-CRTM component.

Currently, in a PC, there are many types of S-CRTM architectures. Two examples are below.

3.3.1.2.1 S-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 is an independent component and each can be updated independent of the other. In this architecture, the BIOS Boot Block is the S-CRTM while the POST BIOS is not, but is a measured component of chain of trust.

End of informative comment.

3.3.1.2.2 S-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 S-CRTM.

End of informative comment.

3.3.1.3 Transferring Control

Prior to transferring control to another entity within the S-RTM environment, an executing entity MUST measure the entity to which it will transfer control.

3.3.2 Integrity Collection, Reporting, and Error Conditions

Start of informative comment:

The Static PCRs are divided into two primary sets. The first set is designated for the Host Platform's pre-OS environment (PCRs[0-6]) and the other designated for the Host Platform's Static OS (PCR[8-15]). PCR[7] may be used in either as determined by the platform manufacturer. The Static pre-OS PCRs provide the Host Platform's initial chain of trust starting from Host Platform Reset. These establish a chain of trust from the S-CRTM through the OS's IPL Code. The definition of the Static OS PCRs is outside the scope of this specification and is the purview of either the specific OS provider or a TCG OS-specific specification, if one exists.

The platform may be designed to allow an operator to indicate to the platform that the platform is not to create measurements. For example, the platform may provide a "BIOS Setup" utility that allows the operator to disallow platform measurements. This setting may be stored into the platform's CMOS, for example. If the operator chooses this option, the platform must be designed to deactivate the TPM along with preventing the BIOS from creating entries into the event log.

The property of the Extend operation makes the set of the integrity measurements taken into each PCR order-dependent. If two measurements, A and B, are extended into a single PCR, the PCR's resulting content will be different if the Extends are performed A then B vs. B then A. For this reason, the order in which the measurements are extended is important because platform applications may "seal" data to the pre-OS PCRs. A seal operation only functions properly with strict adherence to the measurement sequence.

Within each PCR is a list of measurements. The order of the measurements is mostly advisory and some variance is both allowed and expected even with differing platform models from the same manufacturer. However, to make the seal (and other TPM operations) function property, it is important that the sequence of the measurements be consistent between each platform reset when there are no security-related changes. BIOS and other pre-OS software writers must be careful to design the components such that changes to the measurement sequence are not made inadvertently between each platform reset unless the execution or boot sequence actually changes.

End of informative comment.

3.3.2.1 Integrity Collection and Reporting

1. The BIOS MUST be designed to perform integrity measurements of the Host Platform's pre-OS environment per this specification.
2. Integrity collection and reporting 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 configurable) the Host Platform was delivered.
3. The Host Platform MAY provide a method for the owner or operator to prevent the platform from making measurements. If the owner or operator indicates that the Host Platform is not to make measurements, then S-CRTM MUST deactivate the TPM.
4. Integrity measurements by the BIOS that are extended into PCRs[0-6] MUST be done only while the Host Platform is in the Pre-Boot state after starting from an Off state; integrity measurements of the pre-OS environment MUST NOT be extended to the PCRs allocated to the operating system. Any Host Platform configuration change that occurs after the Host Platform transfers control to an operating system, or while resuming from other power states, is the purview of the operating system and measurement of those events MUST be done to the PCRs allocated to the operating system.
5. Integrity measurements made by the platform manufacturer for PCR[7] MAY occur at any time (Pre-Boot or Post-Boot). See Section 3.3.3.8 (PCR[7] – Host Platform Manufacturer Control).
6. If the TPM is deactivated, Events (including EV_SEPARATOR) MUST NOT be logged into the event log.

7. If the TPM is activated, the BIOS measurements and associated event log entries MUST be made, even if the TPM is disabled.
8. While not all events described in Section 11.3.3 (EV_ACTION Event Types) are produced by every platform on every boot, when one is produced, this BIOS MUST create the event indicated in Section 11.3.3 (EV_ACTION Event Types).

3.3.2.2 Error Conditions

3.3.2.2.1 Errors during S-CRTM TPM Initialization

Start of informative comment:

If the S-CRTM is unable to successfully initialize the TPM using the TPM_Startup command, it is possible that later code could successfully issue the command and record false integrity measurements. To prevent this, the S-CRTM takes steps to prevent use of the TPM or platform if it is unable to successfully initialize the TPM. A special case is when the attempt to initialize the TPM results in the TPM being in failure mode; later components will be unable to record false integrity measurements because for most commands the TPM will continue to return TPM_FAILEDSELFTEST.

This section applies for resume from S3 or normal boot.

Figure 6 shows the remediation steps the S-CRTM takes when initializing the TPM fails.

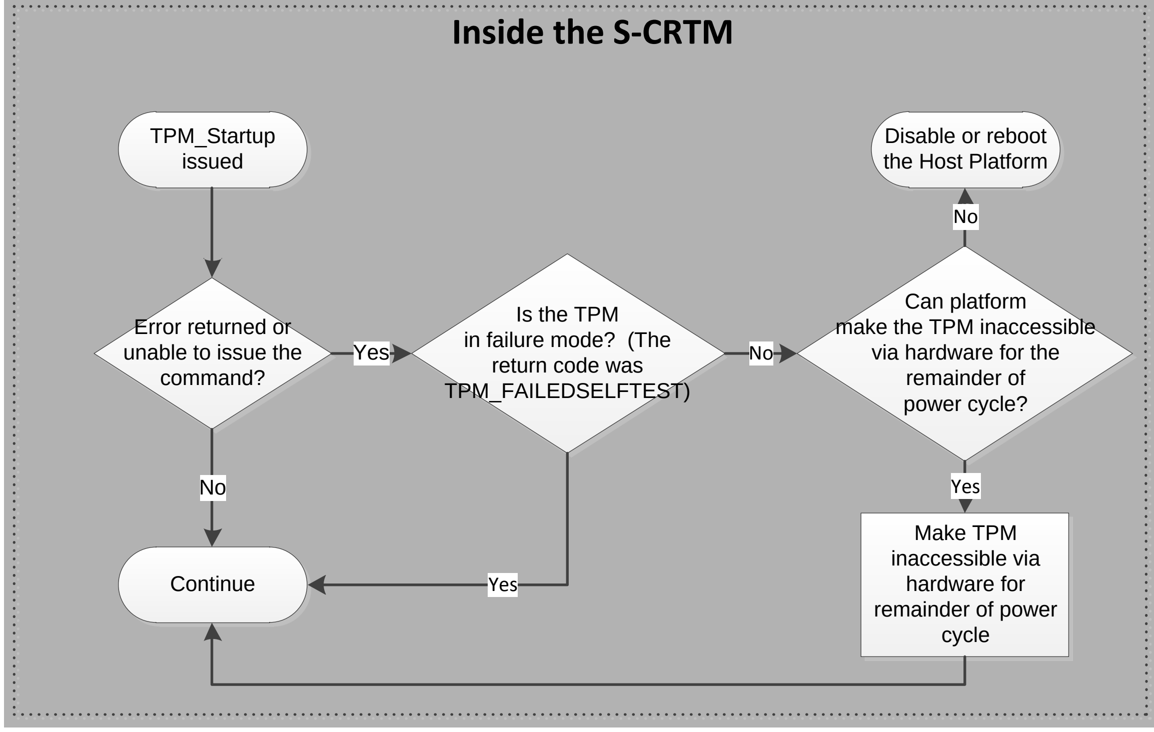
Inside the S-CRTM
TPM_Startup issued
Error returned or unable to issue the command?
Yes
No
Is the TPM in failure mode? (The return code was TPM_FAILEDSELFTEST)
Yes
No
Can platform make the TPM inaccessible via hardware for the remainder of power cycle?
No
Yes
Disable or reboot the Host Platform
Make TPM inaccessible via hardware for remainder of power cycle
Continue

Figure 6: S-CRTM remediation steps when initializing the TPM

End of informative comment.

If the TPM interface is accessible and one of the following situations occur:

1. the S-CRTM is unable to successfully issue the TPM_Startup command, OR
2. the S-CRTM issues the TPM_Startup command and the return result does not equal TPM_SUCCESS or TPM_FAILEDSELFTEST

then the S-CRTM MUST either

1. make the TPM interface inaccessible via hardware for the remainder of the power cycle;
2. reboot the Host Platform;
3. disable the Host Platform; OR
4. perform a vendor-specific action that is equivalent to one of the options above.

3.3.2.2.2 Errors Recording S-RTM Measurements

If the measurement of the S-CRTM, POST BIOS or Embedded Option ROMs cannot be made, the S-RTM MUST be capped by extending the value of 01h to each PCR[0-7], using the EV_SEPARATOR event type per Section 11.3.1 (Event Types). The S-RTM SHOULD log the error event to the event log. If each PCR[0-7] cannot be capped, 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 by performing one of these actions:

1. make the TPM interface inaccessible via hardware for the remainder of the power cycle;
2. reboot the Host Platform;
3. disable the Host Platform; OR
4. perform a vendor-specific action that is equivalent to one of the options above.

3.3.3 PCR Usage

Start of informative comment:

PCR[0:15] 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 seven PCRs are defined for use within the Pre-OS environment (PCR[4] being transition code, which is partially Pre-OS and partially Post-OS). Throughout the BIOS boot process, a log of all executable code and relevant configuration data is created and extended into PCRs as described below.

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.

This section also describes the use of PCRs 16 and 23.

End of informative comment.

PCR Index	PCR Usage
0	S-CRTM, BIOS, Host Platform Extensions, and Embedded Option ROMs
1	Host Platform Configuration
2	Option ROM Code
3	Option ROM Configuration and Data
4	IPL Code (usually the MBR) and Boot Attempts
5	IPL Code Configuration and Data (for use by the IPL Code)
6	State Transitions and Wake Events
7	Host Platform Manufacturer Specific
8-15	Defined for use by the Static OS

16	Debug
23	Application Support

Table 2: Summary of defined PCR usage

3.3.3.1 PCR[0] – S-CRTM, POST BIOS, and Embedded Option ROMs

Start of informative comment:

The S-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 PC Motherboard. Primarily executable code, some fairly stable version identifiers, and configuration data are measured. Configuration data such as ESCD should not be measured as part of this PCR.

All these components are under the control of the manufacturer or its agent.

If for any reason a measurement cannot be made to PCR[0], none of the other S-RTM PCR values can be trusted and therefore are outside the chain of trust. It is therefore necessary to invalidate all Host Platform S-CRTM PCRs as described in Section 3.3.2.2 (Error Conditions).

Because the measurement of the POST is typically done within a resource-constrained environment (for example, the S-CRTM, likely the BIOS Boot Block) the event log corresponding to the Extend event may not 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 (for example, 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 (for example, 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.

Usage of the Event Type EV_POST_CODE

The intent of the event type EV_POST_CODE is to record measurements of components of the Host Platform's transitive trust chain. The imperative is to avoid omitting any portion of the transitive trust chain. If necessary, the event should be used as a catch-all for trust chain components provided by the platform manufacturer, even if the components are not technically "POST code" for the manufacturer's specific implementation.

The normative text below recommends values for the event field for the event type EV_POST_CODE of "POST CODE", "SMM CODE", "ACPI DATA", "BIS CODE", and "Embedded Option ROM". The reason for the recommendation is to promote consistency across implementations by different platform manufacturers. Per the discretion of the platform manufacturer, different values may be used that are relevant for their implementation.

Embedded Option ROMs and PCR[0], PCR[2] and PCR[3]

Manufacturer Controlled Embedded Options ROMs may be Option ROMs that the platform manufacturer physically soldered to the motherboard or Option ROMs whose code is embedded within a firmware image. Embedded Option ROMs are under the control of the platform manufacturer and should not be intended for modification by the platform owner. Their code is measured in PCR[0] using the event EV_POST_CODE because their measurement may be combined with other firmware measurements.

In contrast, Non-Manufacturer Controlled Embedded Option ROMs or Option ROMs in adapter slots may be added, removed, or updated by a platform owner and their code is measured in PCR[2].

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 in PCR[2].

Hidden Option ROM code measurements are always placed in PCR[2]. Option ROM configuration measurements are always placed in PCR[3]. Recording Manufacturer and Non-Manufacturer controlled hidden Option ROM code consistently in PCR[2] and PCR[3], respectively, removes the need for an Option ROM code to know how it was packaged in a system. (For example, as a Manufacturer Controlled Embedded Option ROM versus as an Option ROM in an adapter slot.)

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

PCR[0] typically represents a 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 platform manufacturer-provided 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 set of installed 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 pre-OS and the post-OS state of the platform. This PCR, along with PCR[5], identifies the initial OS loader.

End of informative comment.

Entities that MUST be measured if the TPM is activated:

1. The event type EV_NO_ACTION Specification Event. See Section 11.3.4.1 (Specification Event).
   Note This event is not extended.
2. The S-CRTM's version identifier, using the event type EV_S_CRTM_VERSION. See Section 11.3.1 (Event Types).
3. A variety of entities listed below are measured using the event type EV_POST_CODE. This information MAY be measured as a single combined event or MAY be measured as multiple separate events. The event(s) MUST be recorded using the event type EV_POST_CODE. See Section 11.3.1 (Event Types). If a combined event is measured, the event field SHOULD be the string, "POST CODE" in all caps. If separate events are measured, the event field SHOULD be the string specified below for the entity measured. The entities measured include but are not necessarily limited to the following:
   a. All Host Platform firmware physically bound to the PC Motherboard that is executed by the Host Platform's CPU(s) and is part of the Host Platform's transitive trust chain. If the platform supports entering S2 or S3, this includes the S2 and/or S3 resume code.
      i. POST code (SHOULD use event field string of "POST_CODE" in all caps).
      ii. Embedded SMM code and the code that sets it up (SHOULD use event field string of "SMM CODE" in all caps).
   b. BIS code (excluding the BIS Certificate) (SHOULD use event field string of "BIS CODE" in all caps).
   c. ACPI flash data prior to any modifications. This requirement may be met by either measuring the compressed image in flash or by measuring the in-memory expansion before fix-ups; however, the measurements MUST be made the same way across every boot cycle. (SHOULD use event field string of "ACPI DATA" in all caps.)

   d. Manufacturer Controlled Embedded Option ROMs as a binary image whose release and update is controlled by the Host Platform Manufacturer (SHOULD use event field string of “Embedded Option ROM”).

4. 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., using the event type EV_NONHOST_INFO. See Section 11.3.1 (Event Types).
5. Per Section 3.3.2.2 (Error Conditions), if an error occurs, a value of 01h MUST be extended in PCRs[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 11.3.1 (Event Types).
6. Per Section 8.2.3 (Measuring Boot Events), an EV_SEPARATOR event MUST be measured exactly once for PCRs[0-7] prior to the first invocation of the first INT 19h call. See Section 11.3.1 (Event Types).
7. In some BIOS update scenarios during resume from S2 or S3, per requirements in sections 8.3.2.5 (S2 (Sleep) to S0 (Working)) and 8.3.2.7 (S3 (Sleep) to S0 (Working)), a value of 01h MAY be extended in PCR[0] to invalidate PCR[0].

Entities that MAY be measured if the TPM is activated:

1. The S-CRTM itself using event type EV_S_CRTM_CONTENTS. See Section 11.3.1 (Event Types).

Entities that SHOULD be measured if the TPM is activated:

1. Components within Non-Host Platforms (e.g., firmware not intended to be executed by 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. If measured, this event MUST be recorded using the event type EV_NONHOST_CODE. See Section 11.3.1 (Event Types).

Method for measurement of a compound BIOS:

The S-CRTM performs these measurements as follows:

1. Measure the S-CRTM’s version identifier.
2. Measure the code to which the CRTM is transferring control.
3. 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] again with this reconstructed information.
4. The remaining measurements MAY be performed in any order, except for the EV_SEPARATOR prior to INT 19h invocation, which MUST be last.
5. The specification event is not measured and it MAY be created at any time, but it MUST appear first in the event log. See Section 11.3.4.1 (Specification Event). Note: This event is not extended.

Method for measurement of an integrated BIOS:

The CRTM performs these measurements as follows:

1. Measure the S-CRTM’s version identifier.
2. The remaining measurements MAY be performed in any order, except for the EV_SEPARATOR prior to INT 19h invocation, which MUST be last.
3. The specification event is not measured and it MAY be created at any time, but it MUST appear first in the event log. See Section 11.3.4.1 (Specification Event). Note: This event is not extended.

3.3.3.2 PCR[1] – Host Platform Configuration

Start of informative comment:

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

For performance reasons, some implementations may combine CPU microcode updates with a BIOS POST code image. In this situation, it is acceptable to record CPU microcode updates in PCR[0] as part of an EV_POST_CODE event.

Because platforms may contain a variety of hardware that may or may not be security-relevant for a platform owner's use case, platform manufacturers may allow a platform owner to configure which optional PCR[1] measurements are recorded during the platform boot process. As a result, this section has a large number of entities that may be optionally measured. The platform manufacturer may provide a setup utility that allows the platform owner to choose measurements of interest to record during the boot process. The setup utility may allow a platform owner to select entities whose measurements are optional in this specification to be measured "a la carte" or "all or nothing." Which measurements are currently on or off are measured using the EV_PLATFORM_CONFIG_FLAGS event in a vendor-specific format. The EV_PLATFORM_CONFIG_FLAGS event is intended to be used to only describe events whose measurements in PCR[1] can be toggled on or off; the event data does not contain information about measurements not able to be configured to be measured in PCR[1]. If the platform manufacturer does not permit any configuration of measurements for optional PCR[1] measurements, it does not need to record the EV_PLATFORM_CONFIG_FLAGS event.

An example event data field for EV_PLATFORM_CONFIG_FLAGS for a platform that allows toggling of measurements for only the SMBIOS, BIS Certificate, and ESCD could be the ASCII string "SMBIOS=0;BIS=1;ESCD=0" when the platform is configured to record BIS Certificate information but not the SMBIOS nor the ESCD. If a vendor-specific setup tool was used to also measure ESCD, the event data for EV_PLATFORM_CONFIG_FLAGS would then contain the ASCII string "SMBIOS=0;BIS=1;ESCD=1".

Information about which optional entities are measured by the current boot process is represented in the required "Host Platform Configuration" measurement. Toggling which optional components are measured will always change the value for PCR[1] because the measurement for the EV_PLATFORM_CONFIG_FLAGS will reflect the change.

The Boot Integrity Services (BIS) Certificate may contain information that is privacy-sensitive; thus, recording a measurement of the BIS Certificate is optional. The BIS Certificate may be used by a different boot mechanism on the platform so it is measured in PCR[1] instead of being associated with initial program loader data in PCR[5]. The BIS Certificate may be maintained by the BIOS and built into an SMBIOS structure at boot time. Because other SMBIOS structures are measured in PCR[1], the BIS Certificate is, too.

SMBIOS structures are defined in the external SMBIOS specification. Some example SMBIOS structure values are how many slots exist on the platform for memory and how many slots are currently populated with memory. Many SMBIOS structures exist, but the platform manufacturer should only record security-relevant SMBIOS structures. An example is the system-wide hardware security settings.

CMOS and NVRAM data measured into PCR[1] is placed in the event data field. The expected size of the data is very small. Any data that is security-sensitive, contains dynamic boot data or is dynamic (like the real-time clock) should be omitted. The NVRAM data is the host platform NVRAM data, not the TPM NVRAM.

The EV_ACTION events "User Password Entered", "Administrator Password Entered" and "Password Failure" sound specific to a password being typed but are actually generic to the BIOS user or the BIOS administrator authenticating or failing to authenticate. These events should not be recorded for each password entry attempt made by the operator, but instead should be recorded once per a boot if authentication succeeds or fails based on the platform's password policy for allowed incorrect attempts. For example, a BIOS could be designed to only prompt an administrator for a password if it is less than 15 minutes since the administrator last authenticated during boot by typing the password. For all boots in the following 15 minutes, the BIOS may record the event "Administrator Password Entered" even though the administrator did not physically type a password. An alternative example is that the event could indicate a pre-boot fingerprint reader authenticated the administrator without an actual password having been typed. For scenarios that unseal data based on this PCR's measurement, there may be value in having consistent measurements across boots whether the administrator or a user boots the system. As such, it may be useful for the BIOS to not discriminate between a user or an administrator password being entered by always recording "User Password Entered".

End of informative comment.

Entities that MUST be measured if the TPM is activated:

The following entities MUST always be measured. The platform manufacturer MUST NOT provide BIOS options that permit disabling these measurements:

1. If the BIOS loads a CPU microcode update, it MUST be measured, using event type EV_CPU_MICROCODE. See Section 11.3.1 (Event Types). Exception: Alternatively, CPU microcode updates MAY be measured in PCR[0] as part of a EV_POST_CODE event.
2. If the Host Platform supports selecting of optional PCR[1] measurements, it MUST measure which PCR[1] measurements are currently on or off using the event type EV_PLATFORM_CONFIG_FLAGS. The event data MUST not vary across boot cycles if the set of potential PCR[1] measurements measured does not vary. See the informative text for this section and Section 11.3.1 (Event Types).
3. Prior to entering a ROM-Based Setup Utility, per Section 2.1 (Pre-Boot ROM-Based Setup), if the Host Platform does not unconditionally perform a Host Platform Reset upon exiting a Pre-Boot ROM-based Setup Utility, the platform MUST measure the EV_ACTION event "Entering ROM Based Setup". See Section 11.3.3 (EV_ACTION Event Types). (Note: This is only for ROM-Based Setup Utilities versus entering an Option ROM-Based Setup Utility that is recorded in PCR[3].)
4. Per Section 3.3.2.2 (Error Conditions), if an error occurs, a value of 01h MUST be extended in PCRs[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 11.3.1 (Event Types).
5. Per Section 8.2.3 (Measuring Boot Events), an EV_SEPARATOR event MUST be measured exactly once for PCRs[0-7] prior to the first invocation of the first INT 19h call. See Section 11.3.1 (Event Types).

Entities that MAY be measured if the TPM is activated:

The following entities MAY be measured. Even if measurement of these is provided by the BIOS, the BIOS MAY allow the owner to disable individual measurements or all of the measurements. (For example, disabling of individual measurements could be done using BIOS configuration settings.)

1. SMBIOS structures, using event type EV_EVENT_TAG with the TaggedEventStruct structure data described in Section 11.3.2.3.1 (SMBIOS Structure).
2. BIS Certificate, using event type EV_EVENT_TAG with the TaggedEventStruct structure data described in Section 11.3.2.3.2 (BIS Certificate).
3. POST BIOS-Based ROM strings, using event type EV_EVENT_TAG with the TaggedEventStruct structure data described in Section 11.3.2.3.3 (POST BIOS ROM Strings).
4. ESCD, using event type EV_EVENT_TAG with the TaggedEventStruct structure data described in Section 11.3.2.3.4 (ESCD).
5. Security-relevant CMOS data, using event type EV_EVENT_TAG with the TaggedEventStruct structure data described in Section 11.3.2.3.5 (CMOS). Sensitive data like power on password MUST be omitted from the CMOS data because it is placed unencrypted in the TCG event log.
6. Platform NVRAM data that contains security-relevant user configuration setup options in effect when the platform boots, using event type EV_EVENT_TAG with the TaggedEventStruct structure data described in Section 11.3.2.3.6 (NVRAM). Sensitive data like passwords MUST be omitted from the NVRAM data because it is placed unencrypted in the TCG event log.

7. Table of devices, using event type EV_TABLE_OF_DEVICES described in Section 11.3.1 (Event Types).

   This event 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 11.3.2.3.13 (Host Platform Manufacturer Table of Devices). The BIOS MAY create multiple entries of this event or MAY choose to encapsulate all the data into a single entry.

8. Non-Host Platform configuration information. If the system contains a Non-Host Platform, the BIOS SHOULD use the event type EV_NONHOST_CONFIG to record any security-relevant configuration information or data. See Section 11.3.1 (Event Types).

9. If BIOS user authentication occurred during the boot process, the platform MAY record the EV_ACTION event “User Password Entered”. This could be caused by the user being physically present and typing the password or by BIOS policy determining through other means that the user is currently authenticated during the boot process. See Section 11.3.3 (EV_ACTION Event Types).

10. If BIOS administrator authentication occurred during the boot process, the platform MAY record the EV_ACTION event “Administrator Password Entered”. This could be caused by the administrator being physically present and typing the password or by BIOS policy determining through other means that the administrator is currently authenticated during the boot process. See Section 11.3.3 (EV_ACTION Event Types).

11. If BIOS user or BIOS administrator authentication failure occurs during the boot process, the platform MAY measure the EV_ACTION event “Password Failure”. This could be caused by the operator mistyping the password multiple times, other authentication mechanisms failing, or policy failing to authenticate a BIOS user or the BIOS administrator. See Section 11.3.3 (EV_ACTION Event Types).

12. If the BIOS has detected the platform case was opened or an analogous action occurred, the platform MAY record the EV_ACTION event “Chassis Intrusion”. The event MAY be persistently recorded across platform boots until the BIOS administrator clears the notification. See Section 11.3.3 (EV_ACTION Event Types).

13. If the user altered the boot sequence, the platform MAY record the EV_ACTION event “Boot Sequence User Intervention”. See Section 11.3.3 (EV_ACTION Event Types).

Entities that MUST NOT be measured as part of the above measurements:

1. Values and registers that are automatically updated (e.g., clocks).
2. System-unique information such as asset, serial numbers, etc.
3. The BIS Certificate if it contains privacy-sensitive information.

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, except for the EV_SEPARATOR prior to INT 19h invocation, which MUST be last.
2. Where possible, these measurements SHOULD occur prior to measuring Option ROMs.

3.3.3.3 PCR[2] – Option ROM Code

Start of informative comment:

Option ROMs contained on Host Platform adapters are measured by the BIOS to PCR[2].

Non-Manufacturer Controlled Embedded Option ROMs are embedded Option ROMs that are physically contained on the PC Motherboard (as opposed to an add-in card), but the release and control of any update is not controlled by the Platform Manufacturer. These are measured in PCR[2].

Visible and Hidden Option ROM Code

There may be two portions of Option ROMs: Visible and Hidden. Each is measured in PCR[2].

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

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.

Interpreting Option ROM Measurements

The Option ROM measurements in this specification do not correlate measurements of hidden Option ROM code with the Option ROM that measured the hidden Option ROM code. An evaluator of the measurement log may need to have behavioral knowledge of the Option ROMs on the Host Platform to evaluate the measurements in the log and correlate which measurement is associated with a given Option ROM.

End of informative comment.

The BIOS MUST measure the visible partition of the Option ROM or the Non-Manufacturer Controlled Embedded Option ROM into PCR[2] prior to executing it.

Manufacturer Controlled Embedded Option ROM code measured in PCR[0] as described in PCR[0] requirements SHOULD NOT be measured again in PCR[2] to prevent measuring it twice.

Entities that MUST be measured if the TPM is activated:

1. The portion of the Option ROM that is visible to the BIOS using EV_EVENT_TAG with the TaggedEventStruct structure data containing the OptionROMExecuteStructure structure described in Section 11.3.2.3.7 (Option ROM Execute). This measurement is recorded by the BIOS.
2. The portion of the Option ROM that is not visible to the BIOS using EV_EVENT_TAG with the TaggedEventStruct structure data containing the OptionROMExecuteStructure structure described in Section 11.3.2.3.7 (Option ROM Execute). This measurement is recorded by the Option ROM code.
3. The portion of the Non-Manufacturer Controlled Embedded Option ROM that is visible to the BIOS using EV_EVENT_TAG with the TaggedEventStruct structure data containing the OptionROMExecuteStructure structure described in Section 11.3.2.3.7 (Option ROM Execute). This measurement is recorded by the BIOS.
4. The portion of the Non-Manufacturer Controlled Embedded Option ROM that is not visible to the BIOS using EV_EVENT_TAG with the TaggedEventStruct structure data containing the OptionROMExecuteStructure structure described in Section 11.3.2.3.7 (Option ROM Execute). This measurement is recorded by the Non-Manufacturer Controlled Embedded Option ROM code.
5. The portion of the Manufacturer Controlled Embedded Option ROM that is not visible to the BIOS using EV_EVENT_TAG with the TaggedEventStruct structure data containing the OptionROMExecuteStructure structure described in Section 11.3.2.3.7 (Option ROM Execute). This measurement is recorded by the Manufacturer Controlled Embedded Option ROM code.
6. Per Section 3.3.2.2 (Error Conditions), if an error occurs, a value of 01h MUST be extended in PCRs[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 11.3.1 (Event Types).
7. Per Section 8.2.3 (Measuring Boot Events), an EV_SEPARATOR event MUST be measured exactly once for PCRs[0-7] prior to the first invocation of the first INT 19h call. See Section 11.3.1 (Event Types).

Method for measurement:

1. The BIOS MUST measure the event OptionROMExecute for each Option ROM or Non-Manufacturer Controlled Embedded Option ROM prior to initializing the Option ROM.
2. The BIOS MUST measure the event type EV_POST_CODE for each Manufacturer Controlled Embedded Option ROM (in PCR[0], see Section 3.3.3.1 (PCR[0] – S-CRTM, POST BIOS, and Embedded Option ROMs)) prior to initializing the Manufacturer Controlled Embedded Option ROM.
3. The visible portion of Option ROMs measured in this PCR by the BIOS MAY be measured in any order deemed appropriate by the implementer. However, the order MUST be consistent across boot cycles if the physical location of the Option ROMs is unchanged. (Changing the adapter slot in which an Option ROM is inserted MAY change the order of measurements.) (The BIOS MAY measure all Option ROMs and then initialize them all or MAY measure one and then initialize it before initializing the second, or may use some other variation per the discretion of the manufacturer.)
4. Repeat until all Option ROMs are measured and executed.
5. When initialized, the Option ROMs MUST measure the “hidden” Option ROM code.
6. The Option ROMs MUST measure the “hidden” Option ROM code prior to executing it.
7. The EV_SEPARATOR event prior to INT 19h invocation MUST be measured last.

3.3.3.4 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 trust properties of the Host Platform. Option ROMs or adapters that host Option ROMs perform these measurements.

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

If an Option ROM has a Pre-Boot Setup Utility, per Section 2.1 (Pre-Boot ROM-Based Setup), it needs to record the event type EV_ACTION event “Entering ROM Based Setup” in PCR[3] prior to entering the utility.

If an Option ROM Pre-Boot Setup Utility permits the user to change Option ROM configuration data that is eventually recorded in PCR[3] and the setup utility does not depend on the configuration data recorded in PCR[3], the Option ROM may be designed to permit the configuration changes to occur before the Option ROM records the configuration data in PCR[3]. This may allow the Pre-Boot Setup Utility to avoid needing to perform a Host Platform Reset if the Option ROM configuration is changed.

End of informative comment.

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

Entities that MUST be measured if the TPM is activated:

1. If applicable for the Option ROM or the adapter that hosts the Option ROM, it MUST measure configuration data specific to the device using EV_EVENT_TAG with the TaggedEventStruct structure data containing the OptionROMConfigStructure structure described in Section 11.3.2.3.8 (Option ROM Configuration).
2. If applicable for the Option ROM or the adaptor that hosts the Option ROM, it MUST measure other data, including comments, specific to the device using the event type EV_ACTION with a vendor-specific ASCII “<OpROM Specific non-IPL String>” value as described in Section 11.3.3 (EV_ACTION Event Types) as determined by the device manufacturer.
3. Prior to entering an Option ROM-Based Setup Utility, per Section 2.1 (Pre-Boot ROM-Based Setup), if the Host Platform does not unconditionally perform a Host Platform Reset upon exiting a Pre-Boot ROM-Based Setup Utility, the platform MUST measure the EV_ACTION event “Entering ROM Based

Setup". See Section 11.3.3 (EV_ACTION Event Types). (Note: This is only for Option ROM-Based Setup Utilities versus entering a ROM-Based Setup Utility which is recorded in PCR[1].)

4. Per Section 3.3.2.2 (Error Conditions), if an error occurs, a value of 01h MUST be extended in PCRs[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 11.3.1 (Event Types).
5. Per Section 8.2.3 (Measuring Boot Events), an EV_SEPARATOR event MUST be measured exactly once for PCRs[0-7] prior to the first invocation of the first INT 19h call. See Section 11.3.1 (Event Types).

Entities that MUST NOT be measured as part of the above measurements:

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

Method for measurement:

The Option ROM or Application performs these measurements as follows:

1. Measures the event OptionROMConfigConfiguration.
2. The remaining measurements MAY be performed in any order, except for the EV_SEPARATOR prior to INT 19h invocation, which MUST be last.

3.3.3.5 PCR[4] – Initial Program Loader (IPL) Code and Boot Attempts

Start of informative comment:

Overview

Systems often support the ability to boot from multiple boot devices in priority order. PCR[4] records the process of attempting to boot different hardware paths like from a DVD or a hard drive, what boot devices are attempted, and the IPL Code that is loaded and executed from the device. If boot from one device fails, measurements in PCR[4] record the attempt to boot the next device or boot path. If IPL Code returns control back to the BIOS, each subsequent execution of IPL Code is separately measured.

If boot fails for one device, it often returns execution control back to the BIOS so the BIOS may boot another device. Ultimately the boot process reflected in PCR[4] measurements may contain code from multiple boot devices. The BIOS records events in PCR[4] to demark different boot attempts. When interpreting measurements in PCR[4], verifiers should not disregard code that executed from failed boot attempts. Code that boots first has the ability to record events, and malicious code could record a false demarcation and subsequent events that look like a second device booting.

If the boot device or the IPL Code is not TCG-aware, it may have loaded additional unmeasured code. However, there is a record in PCR[4] showing entry to untrusted code.

The IPL Code (typically the MBR) that BIOS decides to load and measure into PCR[4] may contain code that is part of the OS environment. IPL Code should record code to which it transfers control into PCR[4] or PCRs allocated for the OS. IPL Code should also record additional configuration information it uses into PCR[5] or PCRs allocated for the OS. While this specification does not place requirements on OS loader components or an OS, if the IPL Code does not record measurements of later components, it will break the Static Root of Trust for Measurement.

Conditionally Measuring Which Device Attempts to Boot

Versions of this specification before version 1.21 required measuring which device attempted to boot IPL Code even if the attempt did not execute any code not recorded previously in PCR[0] or PCR[2]. A drawback of this requirement was that inserting unbootable media into a boot device caused the boot measurements in PCR[4] to change even if code loaded from the media was never executed. Also, because the boot attempt event uniquely identified the device, replacement of a faulty device with a like device booting the exact same code would cause the PCR[4] measurement to change. A benefit of the requirement was that the log contained more information about which device booted when attempting to boot from a device that had an event defined in the specification for its device type.

For some hardware, it is possible for the BIOS to determine whether a device is not bootable without executing code that had not been previously recorded in PCR[0] or PCR[2] during the boot attempt. An example is a DVD drive whose BIOS driver is measured in PCR[0]. The boot attempt may use the BIOS driver to determine no DVD media is in the DVD drive and fail the boot attempt. An extended example is a DVD driver measured in PCR[0] or PCR[2] that prompts the user if they would like to boot from the DVD prior to loading code from DVD media; if the user does not press a confirmation key, the boot attempt fails, having run only code previously measured in PCR[0] or PCR[2]. Another extended example is a USB device with a BIOS driver already measured in PCR[0] or PCR[2] that determines the USB media does not contain boot code and aborts the boot attempt, only running previously measured code.

As of revision 1.21, not recording which device attempted to boot may be controlled via a BIOS configuration option or may be fixed by the Platform Manufacturer. If recording which device attempted to boot is disabled due to BIOS configuration or design, the BIOS records the event type EV_OMIT_BOOT_DEVICE_EVENTS in PCR[4]; otherwise, the event is not recorded.

Boot Device Types

Any boot device should be either a BCV device or a BEV device. Booting to a PARTIES partition is a special case of booting to either a BCV or a BEV device.

End of informative comment.

Entities that MUST be measured if the TPM is activated:

1. If the BIOS is configured or designed to not record each device the BIOS attempts to boot, an EV_OMIT_BOOT_DEVICE_EVENTS event MUST be measured once. See Section 11.3.1 (Event Types).
2. The BIOS MUST record the EV_ACTION event "Calling INT 19h". See Section 11.3.3 (EV_ACTION Event Types).
3. Per Section 8.2.3 (Measuring Boot Events), an EV_SEPARATOR event MUST be measured exactly once for PCRs[0-7] prior to the first invocation of the first INT 19h call. See Section 11.3.1 (Event Types).
4. When the BIOS transfers control to the INT 19h handler, the BIOS MAY record the EV_ACTION event "Returned INT 19h". See Section 11.3.3 (EV_ACTION Event Types).
5. For each boot attempt, if the EV_OMIT_BOOT_DEVICE_EVENTS event was not recorded, the BIOS MUST record the following EV_ACTION string per Section 11.3.3 (EV_ACTION Event Types) when attempting to boot the corresponding type of device:
    a. "Booting BCV Device s"
    b. "Booting BEV Device s"
    c. "Booting to Parties N"
6. Each IPL that is attempted and executed, using the event EV_IPL.See Section 11.3.1 (Event Types). See Section 8.2.3 (Measuring Boot Events) for specific details for how measurements are done for different boot devices and if applicable corresponding PCR[5] measurements.
7. Additional pre-OS environment code that is loaded by the IPL Code using event EV_COMPACT_HASH. See Section 11.3.1 (Event Types).
8. If a boot device or IPL Code return control back to the BIOS via INT 18h, the BIOS MUST record the EV_ACTION event "Returned via INT 18h". See Section 11.3.3 (EV_ACTION Event Types).
9. If the BIOS receives control back from a failed boot device or IPL via a mechanism other than INT 18h, the BIOS MUST record the EV_ACTION event "Returned INT 19h". See Section 11.3.3 (EV_ACTION Event Types).
10. Per Section 3.3.2.2 (Error Conditions), if an error occurs, a value of 01h MUST be extended in PCRs[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 11.3.1 (Event Types).

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).
2. For each boot attempt, if the EV_OMIT_BOOT_DEVICE_EVENTS event was recorded, the BIOS MUST NOT record the following EV_ACTION string per Section 11.3.3 (EV_ACTION Event Types) when attempting to boot a device:
    a. "Booting BCV Device s"
    b. "Booting BEV Device s"
    c. "Booting to Parties N"

Method for measurement:

This section provides only a short overview of the actions required. See Section 8.2.3 (Measuring Boot Events) for specific details.

Across boot cycles to the same boot device(s) connected in the same way, BIOS measurements into PCR[4] MUST be measured in the same sequence every time. (Note: This means booting the same system twice will result in the same measurements.)

The BIOS performs these steps as follows:

1. Measure EV_OMIT_BOOT_DEVICE_EVENTS if the BIOS is designed or configured to not record which devices it attempts to boot.
2. Measure EV_ACTION event “Calling INT 19h”.
3. Measure EV_SEPARATOR in PCRs[0-7].
4. Measure EV_ACTION event “Returned INT 19h”.
5. Measure EV_ACTION events “Booting BCV Device s”, “Booting BEV Device s” or “Booting to Parties N” if EV_OMIT_BOOT_DEVICE_EVENTS was not measured and the boot device class corresponds to a class boot attempt events are defined for.
6. Measure the IPL Code before transferring control to it. (Note: Some implementations may load IPL Code, but decide not execute it. An example is a BIOS designed to read from a USB device to determine whether it is bootable and conditionally run code from the device if it is. If the USB device is not bootable, the BIOS does not measure it because it never transfers control to it.)
7. Conditionally measure the IPL configuration data in PCR[5] if appropriate for the class of device that is booted per Section 8.2.3 (Measuring Boot Events).
8. IPL Code SHOULD measure additional IPL Code to which it transfers control into PCR[4] or into PCRs reserved for the OS.
9. IPL Code SHOULD measure additional configuration data into PCR[5] or into PCRs reserved for the OS.
10. If control returns to the BIOS, measure the returning event as EV_ACTION event “Returned via INT 18h” OR “Returned INT 19h”, depending on how the device returns control to the BIOS.
11. For additional boot devices or paths, go to Step 5.

3.3.3.6 PCR[5] – IPL Configuration and Data

Start of informative comment:

The IPL Code may have configuration or other data that is relevant to the trust properties of the Host Platform. For some specific situations documented in this specification, the BIOS will record the IPL Data or configuration information. In other cases, IPL Code or Option ROM code itself will record its configuration or other data. In all cases, the configuration information is stable across multiple boot cycles if the boot proceeds in the same manner with the same trust properties. Transient information like time is not recorded in this PCR.

Information measured into this PCR by the BIOS is security-relevant data associated with booting the device. An example is data embedded within the IPL Code such as the disk geometry within the MBR.

An example of IPL Code recording configuration data is IPL Code that allows the selection of alternate boot partitions. The partition selection information would be measured in this PCR by the IPL Code.

Another example of an Option ROM recording configuration data is PXE code measuring partition data it loaded using the EV_ACTION event and encoding the data as an ASCII string.

End of informative comment.

Entities that MUST be measured if the TPM is activated:

1. All relevant IPL configuration data, using the event EV_COMPACT_HASH. See Section 11.3.1 (Event Types).
2. Security-relevant data contained within the IPL Code (e.g., disk geometry), using the event EV_IPL_PARTITION_DATA. See Section 11.3.1 (Event Types) for what to place in the event data field.
3. If applicable, when booting an Option ROM or an adaptor that hosts an Option ROM, it MUST measure other data, including comments, specific to the device using the event type EV_ACTION with a vendor-specific ASCII "<OpROM Specific IPL String>" value as described in Section 11.3.3 (EV_ACTION Event Types) as determined by the device manufacturer.
4. Per Section 8.2.3 (Measuring Boot Events), an EV_SEPARATOR event MUST be measured exactly once for PCRs[0-7] prior to the first invocation of the first INT 19h call. See Section 11.3.1 (Event Types).
5. Per Section 3.3.2.2 (Error Conditions), if an error occurs, a value of 01h MUST be extended in PCRs[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 11.3.1 (Event Types).

Method for measurement:

1. The EV_SEPARATOR event prior to INT 19h invocation.
2. The BIOS measures the static data such as disk geometry.
3. The IPL Code measures all relevant IPL configuration data per its defined events.

3.3.3.7 PCR[6] – State Transition and Wake Events

Start of informative comment:

PCR[6] may contain measurements made by the Platform Manufacturer during boot and later by an OS. Platform Manufacturers may use the PCR to record what turned on or woke up the platform. Operating systems may use the PCR to record state transitions like entering or waking from sleep.

BIOS Measurements

Initial Host Platform bootstrap when resuming from an Off state are one-time events from the perspective of the RTM and the chain of trust.

This specification defines the events the Platform Manufacturer may measure into this PCR. For platforms that always use the same process for booting regardless of the wake source, measuring the wake source is optional. For platforms whose boot order varies depending on the wake source (as stated below), the Platform Manufacturer must record the wake source.

OS Measurements

Other than resume from an Off state, all other resume conditions (for example, resume from S3) 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 an Off state, it is best to leave measuring these events to the operating system. State transition measurements recorded by the OS are OS-specific and are outside the scope of this specification.

Differentiating BIOS from OS Measurements

The EV_SEPARATOR that will be measured into this PCR prior to turning control of the Host Platform to the OS will provide the delimiter between the wake event measured into this PCR by the Pre-Boot components and those that have been measured into this PCR by the OS.

End of informative comment.

Entities that MUST be measured if the TPM is activated:

1. If the platform boot process does vary depending on the wake source, the BIOS MUST measure the EV_ACTION event "Wake Event n", where the value of 'n' is the wake source. For example, wake source zero would get the string value "Wake Event 0". See Section 11.3.3 (EV_ACTION Event Types).
2. If the platform boot process does not vary depending on the wake source, the BIOS MAY measure the EV_ACTION event "Wake Event n", where the value of 'n' is the wake source. See Section 11.3.3 (EV_ACTION Event Types).
3. Per Section 8.2.3 (Measuring Boot Events), an EV_SEPARATOR event MUST be measured exactly once for PCRs[0-7] prior to the first invocation of the first INT 19h call. See Section 11.3.1 (Event Types).
4. Per Section 3.3.2.2 (Error Conditions), if an error occurs, a value of 01h MUST be extended in PCRs[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 11.3.1 (Event Types).

Entities that MUST NOT be measured by the BIOS:

1. Resuming from S1 through S3.

Method for measurement:

1. If implemented, the BIOS measures the EV_ACTION event "Wake Event n".
2. The EV_SEPARATOR event prior to INT 19h invocation MUST be the last BIOS measurement in the PCR.

3.3.3.8 PCR[7] – Host Platform Manufacturer Control

Start of informative comment:

This PCR is reserved for use by the Host Platform manufacturer. This allows for Host Platform Manufacturer-specific applications that operate within the pre-OS or OS environment to use this PCR. The use of this PCR may not be common across different Host Platform manufacturers and even across different Host Platform models within the same Host Platform manufacturer.

It is anticipated the TCG event log used during Pre-OS environment may be copied and managed by TCG-capable operating systems. Additional entries made to the Pre-OS TCG event log after the OS is managing its own copy of the measurement log may be ignored by the OS. This specification does not define a mechanism for a platform manufacturer to add entries to an OS-managed copy of the TCG event log after the OS has started.

It is anticipated that operating systems may have their own TPM driver. This specification does not define a mechanism for a platform manufacturer to send commands to the TPM after the firmware launches an operating system. Therefore, a platform manufacturer may have difficulty extending PCR[7] without interfering with OS management of the TPM.

A platform manufacturer could collaborate with an OS vendor to agree on a platform-specific use of PCR[7] where both the platform manufacturer agree and coordinate the way commands for extending PCR[7] are sent to the TPM and events are logged.

In theory, it is possible for a platform manufacturer to devise a mechanism to extend PCR[7] without interfering with the OS management of the TPM for manufacturer-specific purposes. Therefore, generic operating systems should not expect PCR[7] to always match the log entries in its OS-managed TCG event log.

End of informative comment.

1. The Host Platform Manufacturer MAY define the purpose of this PCR.

2. If the Platform Manufacturer extends PCR[7] during the Pre-OS environment, it MUST record a corresponding log entry.
3. If the Platform Manufacturer extends PCR[7] during the OS environment, it MUST NOT record a corresponding log entry in the Pre-OS TCG event log. However, it MAY record an event by collaborating with the OS to place an event in an event log managed by the OS.
4. The operating system and user applications SHOULD NOT use this PCR for sealing or attestation without knowing manufacturer-specific information about its usage.

Entities that MUST be measured if the TPM is activated:

1. Per Section 3.3.2.2 (Error Conditions), if an error occurs, a value of 01h MUST be extended in PCRs[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 11.3.1 (Event Types).
2. Per Section 8.2.3 (Measuring Boot Events), an EV_SEPARATOR event MUST be measured exactly once for PCRs[0-7] prior to the first invocation of the first INT 19h call. See Section 11.3.1 (Event Types).

3.3.3.9 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.

Because the PCR is not intended for use with sealing or attestation, measurements for it should not be recorded in the event log.

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. On platforms targeted for user, final or production environments, if the firmware does extend PCR[16], it MUST NOT record the event in the event log. (Clarification: Basic firmware functions like TCG_HashLogExtendEvent and TCG_CompactHashLogExtendEvent will record measurements for PCR[16] in the event log. Firmware that does extend PCR[16] MUST NOT use these basic functions to avoid placing an event in the measurement log.)
4. On platforms targeted for user, final or production environments, if an application does extend PCR[16], it SHOULD NOT record the event in the event log.

3.3.3.10 PCR[23] – Application Support

Start of informative comment:

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

The PCR value is not preserved across S3/Resume cycles because it is a resettable PCR.

End of informative comment.

1. The operating system defines the purpose of this PCR and MAY reset and use it at any time.

3.3.4 Localities Assigned to the D-RTM and Transitive Trust Chain

3.3.4.1 Concepts

Start of informative comment:

The Host Platform Manufacturer and Dynamic OS define the usage of Localities 1–4 and PCRs 17–22. The usage of Locality 1–4 is beyond the scope of this specification. Generally, the Host Platform is responsible for maintaining protections of the PCRs based on their associated Locality by controlling access to the TPM_HASH_START command memory address. 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 OS and application developers.

PCR Index	**Usage**	**pcrReset**	**pcrResetLocal for Locality 4, 3, 2, 1, 0**	**pcrExtendLocal for Locality 4, 3, 2, 1, 0**
0 – 15	Static RTM	0	0,0,0,0,0	1,1,1,1,1
16	Debug	1	1,1,1,1,1	1,1,1,1,1
17	Dynamic CRTM	1	1,0,0,0,0	1,1,1,0,0[2]
18	Dynamic RTM	1	1,0,0,0,0	1,1,1,0,0
19	Dynamic RTM	1	1,0,0,0,0	0,1,1,0,0
20	Dynamic RTM	1	1,0,1,0,0	0,1,1,1,0
21	Dynamic RTM	1	0,0,1,0,0	0,0,1,0,0
22	Dynamic RTM	1	0,0,1,0,0	0,0,1,0,0
23	Application Specific	1	1,1,1,1,1	1,1,1,1,1

Table 3: PCR usage

End of informative comment.

3.3.5 Dynamic Core RTM (D-CRTM)

The D-CRTM 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. This is the executable code of the Dynamic RTM. 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-RTM, the value of the D-CRTM's transitive trust chain does not depend on the S-RTM's transitive trust chain.

[2] See the TPM Interface Specification Section 8.1 for exceptions and restrictions to this behavior.

3.3.6 Locality 1– 3

3.3.6.1 Integrity Collecting and Reporting

The method of collecting and reporting PCRs[17-23] is beyond the scope of this specification and is the purview of the Host Platform's hardware and the Dynamic OS that supports the D-RTM.

3.3.6.2 PCR Usage

The usage of PCRs[17-22] is beyond the scope of this specification and is the purview of the Dynamic OS.

3.3.6.3 Protection

Start of informative comment:

How or if a Platform Manufacturer provides access to the address ranges for Localities 1 through 3 is out of scope for this specification.

End of informative comment.

The Host Platform MAY provide protections to enforce the Locality messages from the source of the commands to the TPM.

3.3.7 Locality 4

3.3.7.1 Concepts

Start of informative comment:

The use of Locality 4 is restricted to trusted hardware on the Host Platform. The Host Platform protects the address ranges for Locality 4. Even the Dynamic OS cannot access Locality 4.

The TPM_PCR_RESET command for PCR[17] and the TPM_HASH_* commands require Locality 4. This protects PCR[17] from being reset except by trusted hardware associated with the D-CRTM.

Some platforms and some TPMs may not permit sending commands using the Locality 4 address range even by trusted hardware and may only permit use of the TPM_HASH_* commands for Locality 4.

End of informative comment.

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

3.3.7.2 Integrity Collection and Reporting

How this is done is Host Platform implementation-specific.

3.3.7.3 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.

3.3.7.4 Protection

The Host Platform MUST provide protections to ensure that the entire Locality 4 address range is accessible only by the D-CRTM.

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. A limited amount of resources 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 the TPM must provide. This value is based upon the anticipated usages of this area at the time the specifications are written. It is expected that the operating system and its applications will want at least 512 bytes 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.

Caveat: The calculation and resulting value for NV Storage used below represents an estimate. TPM and Host Platform manufacturers are encouraged to consult with 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 below may not be appropriate for all environments and uses.

End of informative comment.

1. After all the NV RAM storage requirements of the Host Platform manufacturer are satisfied, the Host Platform MUST provide to the static OS or dynamic OS at least 512 bytes total of NV RAM storage.

4.2 NV Storage Indexes

4.2.1 TPM Main Specification Reserved Indexes

Start of informative comment:

It is important, and in some cases necessary, for Host Platform software to obtain Host Platform-related TCG certificates. The TPM and Host Platform manufacturer may (but not necessarily) create these certificates. 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 no compromise of the security properties of the TPM or the Host Platform). However, some (for example, the EK Certificate) do contain privacy-sensitive information—i.e., 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 these certificates. One of the NV Storage attributes is TPM_NV_PER_OWNERREAD, which means only the "current" owner has the authorization to read the specified NV Storage area. However, if the "D" bit attribute is not set, 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 Version 1.2, Part 2 Structures, Section 19.1.2, the following indexes are predefined 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 the TPM manufacturer and/or the Host Platform Manufacturer provide a certificate as defined in Section 7 (TCG Certificates and Verification of a Platform), they MUST define the index using the predefined and reserved index and set the data to the indicated certificate using the format specified in Section 7. The TPM manufacturer and/or Host Platform manufacturer MAY define these indices with the “D” bit attribute set. If the “D” bit attribute is not set, platform manufacturers SHOULD provide guidance to owners to retain these values or replace them with corresponding 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 following indexes are reserved for the indicated entity.

Platform Manufacturers implementing TCG Physical Presence Interface Specification Version 2.0 may need an NV index to store persistent values in the TPM. One of the indices reserved in the TPM 1.2 Structures Specification for the PC Client platform is designated for this purpose.

A best practice for platform user indices (created by the OS or applications) associated with the TPM operator is to use owner permission so the indices are cleared when the TPM is cleared.

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.

Value Range	**Entity Name**
0x0000F000 - 0x0000FFFF	Defined in the TPM Main Specification
0x20010000 - 0x2001FFFF	Reserved for OS usage
0x40010000 - 0x4001FFFF	Reserved for Platform Manufacturer usage
0x50010000	Defined for Platform Manufacturer persistent storage of TPM Management Flags (see the Physical Presence Interface Specification for more information)
0x50010001 - 0x5001FFFF	Reserved for Platform Manufacturer usage

Table 4: NV RAM index values

The Host Platform manufacturer MUST SET the TPM’s permanent flag nvLocked to ensure that NV Storage attributes are enforced.

5. General Purpose I/O (GPIO)

Start of informative comment:

General purpose I/O (GPIO) provides an optional interface between the TPM's command interface and an external device. The actual use and protocol of the signal is implementation-specific and is not specified by the TCG. Only a single GPIO pin is currently defined and it is optional.

The TPM's command interface accesses the GPIO pins using the NV Storage interface. This is much like a "memory-mapped" I/O in other architectures. The TPM Main Specification reserves 256 indices for this purpose tagged TPM_NV_INDEX_GPIO_xx where xx is the range 00-FF. The platform-specific specification may define each index and its association with a specific GPIO pin and that pin's purpose. The PC Client TPM Interface Specification only specifies the routing of the GPIO index to the GPIO pin. It is the purview of the PC Client Implementation Specifications to specify the routing to the specific device. In general, this is done by mapping the data sent in the TPM_NV_WriteValue[3] and TPM_NV_WriteValueAuth commands' 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" 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. If the TPM Owner is removed, the area returns to undefined and must be defined again before use. The reason for this behavior is that the new TPM owner may have different security and privacy requirements for this GPIO.

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

Note to implementers:

Careful examination of the TPM_NV_ATTRIBUTES reveals that if none of the read or write permission fields (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.

Note that the pin-out specified in the PC Client Interface Specification Section 14.1 TPM Packaging is only recommended and is not mandatory. TPMs are allowed to be implemented using any packaging. However, if this packaging is chosen, the pins, including the location of the GPIO pins, are mandatory. If this packaging is not used, the TPM manufacturer must provide documentation to the platform manufacturer indicating which pin is used as a GPIO pin. For ease in documentation, regardless of whether the TPM implements the recommended packages or uses their own, the designation of this pin will be GPIO-Express-00.

[Note: The above is a copy of Section 6 General Purpose I/O 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.]

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 Trusted Configuration Space for PCI Express ECN. 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.

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

Regarding the Trusted Configuration Space for PCI Express ECN, 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 remain 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 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*.

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.
2. The following text applies 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.
    a. The GPIO-Express-00 pin MUST be connected to the PCI Express TCS_EN pin of the PCI Express Root Complex as described in the Trusted Configuration Space for PCI Express ECN.
    b. The Host Platform MUST be designed such that only the GPIO-Express-00 pin is able to drive the TCS_EN pin.
    c. The TCS_EN signal is active low. (For example, 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.)
    d. The Host Platform MUST be designed such that the default state of the TCS_EN pin is inactive.
    e. 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.
    f. The GPIO-Express-00 bit SHALL be the least significant bit of TPM_NV_INDEX_GPIO_00.
    g. The connection from the TPM's GPIO-Express-00 bit to the GPIO-Express-00 pin MUST be such that:
        i. Writing a '1' to the GPIO-Express-00 bit results in holding the GPIO-Express-00 pin to a high state.
        ii. Writing a '0' to the GPIO-Express-00 bit results in holding the GPIO-Express-00 pin to a low state.

    iii. Reading the GPIO-Express-00 bit returns the current state of the GPIO-Express-00 pin.

h. 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 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 per the TPM Main Specification.

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. The BIOS Recovery Mode may not have the capability to continue the S-RTM measurement chain.

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. Use of hardware to disable the TPM until the next TPM_Init; or,
2. Modifications to CRTM recovery code, which would extend the value 01h to the Pre-Boot PCRs[0-7] per Section 3.3.2.2; or,
3. Deactivation of the TPM by calling TPM_Startup (startupType =TPM_ST_ DEACTIVATED); or,
4. Measuring of components to maintain the S-RTM for BIOS Recovery Mode components; or,
5. Unconditionally performing a Host Platform Reset upon completion of BIOS Recovery Code.

End of informative comment.

It MUST NOT be possible for a BIOS Recovery Mode to allow impersonation of another boot state as represented by the S-RTM. This applies to the values in the PCRs[0-7].

6.2 Flash Maintenance

Start of informative comment:

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

End of informative comment.

1. The Platform Manufacturer MUST control the update, modification and maintenance of the CRTM within the MAE. See Section 1.2.2 (Immutable).
2. The MAE is vendor-specific and MUST provide protections for the TBB and S-CRTM.
3. At the completion of the update process, the reset vector MUST point to the one and only one S-CRTM that is controlled by the MAE.
4. At the completion of the update process, the execution MUST begin at the S-CRTM designated by the platform manufacturer.
5. If a BIOS update is installed that modifies the S-CRTM, the BIOS update process MUST unconditionally transition the platform to the Off state or reboot.

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.

Start of 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 led 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 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.

7.1.1 Host Platform Certificate Contents

Trust in trusted processes MAY be expressed in the Host Platform Certificate (e.g., the S-CRTM, the D-CRTM, etc.).

7.2 Host Platform Conformance Certificate

Distribution is manufacturer-controlled.

7.3 Validation of Objects Using a Validation Certificate

Start of informative comment

This section is deprecated in version 1.21 of this specification.

Start of deprecated section:

A Validation Certificate provides certified and trusted information about the 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 measurement 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.

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 below according to the "Plug and Play BIOS Specification."

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 1st segment included in Validation Certificate hash
	WORD	Varies	Length-1 of 1st 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

Table 5: Validation Certificate Header

End of informative comment.

End of deprecated section.

7.4 Storing Certificate in TPM NV Storage

Start of 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 manufacturer's website, 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 with 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

Start of informative comment

These definitions are, in some ways, parallel to the TPM_STRUCTURE_TAG defined in the TPM Main Specification Version 1.2. The purview of these tags are within the PC Client. These structures are never processed by the TPM itself.

End of informative comment.

7.4.1.1 Helper Definitions

Name	Value	Description
TCG_PCCLIENT_STRUCTURE_TAG	UINT16	Identifies size of the tag

Table 6: TCG_PCCLIENT_STRUCTURE_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 determine which platform class this structure belongs to in order to properly parse it.

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

Table 7: Structure Tag Definitions

7.4.2 TPM_CERT_TYPE

Start of informative comment:

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

End of informative comment.

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.

Table 8: Type of certificate representation

7.4.3 TCG_CERT_FLAGS

Start of informative comment:

This flag information could be used in the future to describe 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

Start of informative comment:

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

End of informative comment.

Definition

```
typedef struct tdTCG_PCCLIENT_STORED_CERT {
   TCG_PCCLIENT_STRUCTURE_TAG tag;
   BYTE                       certType;
   UINT16                     certSize;
   BYTE[certSize]             cert;
} TCG_PCCLIENT_STORED_CERT;
```

Type	Name	Description
TCG_PCCLIENT_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[certSize]	cert	The certificate itself. If certType is TCG_FULL_CERT, then this is a TCG_FULL_CERT structure. If certType is TCG_PARTIAL_SMALL_CERT, then this is a TCG_PARTIAL_SMALL_CERT structure.

Table 9: Stored Certificate Structure

7.4.5 TCG_FULL_CERT

Start of 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;
   BYTE[]                     cert;
} TCG_FULL_CERT;
```

Type	Name	Description
TCG_PCCLIENT_STRUCTURE_TAG	tag	MUST be TCG_TAG_PCCLIENT_FULL_CERT
BYTE[ ]	cert	The entire certificate. (This size may be calculated from the TCG_PCCLIENT_STORED_CERT structure's certSize field.)

Table 10: Full Certificate Structure

7.4.6 TCG_PARTIAL_SMALL_CERT

Start of informative comment:

While storing full certificates might be convenient for 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 be stored within the limited NV Storage area. The TPM Main Specification Version 1.2 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 latter 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, the 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 2,048-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 non-volatile 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. This 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, website 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 *nvStore* 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 the utility would encrypt the entire certificate and store it on the disk. It 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;
   BYTE                       certType;
   UINT16                     certFlags;
   BYTE[4]                    certID;
   UINT16                     signatureSize;
   BYTE[signatureSize]        signature;
   UINT16                     additionalDataSize;
   BYTE[additionalDataSize]   additionalData;      // (E.g., serial #)
} TCG_PARTIAL_SMALL_CERT;
```

Type	Name	Description
TPM_STRUCTURE_TAG	Tag	MUST be TCG_TAG_PCCLIENT_PART_SMALL_CERT
BYTE	certType	MUST be TCG_PARTIAL_SMALL_CERT
UINT16	certFlags	MUST be zero
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 website
UINT16	signaturesSize	The size of the signature in bytes
BYTE[signaturesSize]	signature	The signature element of the certificate
UINT16	additionalDataSize	The size of the additional data
BYTE[additionalDataSize]	additionalData	Additional data necessary to build the cert, usage, and size based on the selected template

Table 11: Partial Certificate Structure

8. State Transitions

8.1 Architecture and Definitions

Start of informative comment:

A handoff to an operating system generally occurs after BIOS has completed its initialization and testing of the Host Platform hardware. As defined in Section 1.2.13 (Boot State Transition), the event that marks the transition from the Pre-Boot state to the Post-Boot state on the Host Platform is the first invocation by the BIOS POST code of INT 19h, or an equivalent event.

As stated in Section 6.5 of the BIOS Boot Specification, by the time the INT 19h handler code gets invoked, the BIOS has found and initialized all the IPL devices on the Host Platform for this boot cycle and has built a priority-ordered list of those IPL devices. Each entry in the list includes a pointer to the IPL Code that is capable of booting an operating system from that device.

The BIOS INT 19h handler code starts at the top of the prioritized list and simply attempts to boot an operating system from each IPL device, in turn, by loading and executing the IPL Code on that device.

If the IPL Code on a device fails to boot an operating system, it invokes an INT 18h, or equivalent.

Measuring and Logging Events in the Pre-Boot to Post-Boot Transition

TCG-enabled BIOS must measure and log the events described in the previous section of this Informative comment. In general, the BIOS INT 19h handler code must, without exception, measure into PCR [4] the event of loading and executing IPL Code from an IPL device; this measurement will automatically result in the proper entry being made into the Event Log. The BIOS must not hash any data areas when it measures this event.

If the IPL Code on an IPL device returns back to the BIOS through INT 18h or INT 19h, that event must be measured.

And, as stated in Section 3.3.3.5 (PCR[4] – Initial Program Loader (IPL) Code) of this specification, if IPL Code returns control back to the BIOS, each subsequent execution of IPL Code must be separately measured.

End of informative comment.

8.2 Procedure for Transitioning from Pre-Boot to Post-Boot

Start of informative comment:

In order to measure the transition from the Pre-Boot state to the Post-Boot state, a number of steps need to be performed. This section of the specification will outline and describe 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 the chain of measurements of the Host Platform boot process is contiguous. One of the important events that need 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 code 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 PCR[5]. PCR[5] may be utilized and extended by any boot loader for variable data.

End of informative comment.

8.2.3 Measuring Boot Events

Start of informative comment:

An Event Log is provided in memory for a challenger to make a determination of the state of trust of the Host Platform. The BIOS functions designed for measuring boot events automatically create Event Log entries.

The measurements defined for PCR[4] and PCR[5] are for a platform that successfully boots some IPL Code. The measurements for a platform that fails to boot are not interesting; therefore, failed boot is not specified.

Example 1: Sequence of measuring boot events recording attempts to boot to devices

1. BIOS measures EV_ACTION index 0 "Calling INT 19h" in PCR[4].
2. BIOS measures EV_SEPARATOR in PCRs[0-7].
3. BIOS code invokes the INT 19h handler.
4. INT 19h handler code measures EV_ACTION index 1 "Returned INT 19h" in PCR[4].
5. INT 19h handler points to start of IPL device sequence list.
6. INT 19h handler selects next IPL device from list.
7. If no more IPL devices:
    a. Issue INT 19h and go to step 4, OR
    b. Issue INT 18h, measure EV_ACTION index 2, "Return via INT 18h" in PCR[4] If all devices have been processed, go to step 5; otherwise, if remaining devices, go to step 6, OR
    c. Go to step 4.
8. If leaving BIOS control to check for IPL Code on the device, measure the appropriate EV_ACTION "Booting …" event in PCR[4] if the boot device is a BCV, BEV or PARTIES partition.
9. Check the IPL device for existence of IPL Code; if no bootable IPL Code is found:
    a. Issue INT 19h and go to step 4, OR
    b. Issue INT 18h, measure EV_ACTION index 2, "Return via INT 18h" in PCR[4]. If all devices have been processed, go to step 5; otherwise, if remaining devices, go to step 6, OR
    c. Go to step 4.
10. If not measured in step 8, measure the appropriate EV_ACTION "Booting…" event in PCR[4] if the boot device is a BCV, BEV or PARTIES partition.
11. Measure IPL Code in PCR[4].
12. If applicable, measure partition data in PCR[5].

13. Execute IPL Code.
14. The IPL Code may measure additional IPL Code in PCR[4].
15. The IPL Code may measure additional configuration data in PCR[5].
16. End of the boot process.

Example 2: Sequence of measuring boot events not recording attempts to boot to devices

1. BIOS measures EV_OMIT_BOOT_DEVICE_EVENTS in PCR[4].
2. BIOS measures EV_ACTION index 0 “Calling INT 19h” in PCR[4].
3. BIOS measures EV_SEPARATOR in PCRs[0-7].
4. BIOS code invokes the INT 19h handler.
5. INT 19h handler code measures EV_ACTION index 1 “Returned INT 19h” in PCR[4].
6. INT 19h handler points to start of IPL device sequence list.
7. INT 19h handler selects next IPL device from list.
8. If no more IPL devices:
    a. Issue INT 19h and go to step 5, OR
    b. Issue INT 18h, measure EV_ACTION index 2, “Return via INT 18h” in PCR[4]. If all devices have been processed, go to step 6; otherwise, if remaining devices, go to step 7, OR
    c. Go to step 5.
9. Check the IPL device for existence of IPL Code.
10. If no bootable IPL Code:
    a. Issue INT 19h and go to step 5, OR
    b. Issue INT 18h, measure EV_ACTION index 2, “Return via INT 18h” in PCR[4]. If all devices have been processed, go to step 6: otherwise, if remaining devices, go to step 7, OR
    c. Go to step 5.
11. Measure IPL Code in PCR[4].
12. If applicable, measure partition data in PCR[5].
13. Execute IPL Code.
14. The IPL Code may measure additional IPL Code in PCR[4].
15. The IPL Code may measure additional configuration data in PCR[5].
16. End of the boot process.

End of informative comment.

8.2.4 Passing Control of the TPM from Pre-Boot to Post-Boot

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 Post-Boot and the Pre-Boot environments for use and access of the TPM.

The INT 1Ah interface provides for a solution to this problem through function TCG_ShutdownPreBootInterface. 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. IPL Code should not return control back to the BIOS after calling TCG_ShutdownPreBootInterface because the BIOS may lose its ability to measure additional events once the method completes.

If the operating system is to use the transitive trust chain beyond the measurements indicated in this specification, it is expected that the IPL 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.

If the IPL Code is booting an OS, it MAY measure components into PCR[>7].

After IPL Code has called TCG_ShutdownPreBootInterface, the BIOS MAY NOT record additional measurements.

8.2.4.1 **Hard Disk Device or Hard Disk-Like Devices**

Start of informative comment:

This section defines the default behavior for how a BIOS should record the measurements of a Master Boot Record (MBR). This default guidance should be used unless a subsequent section has more specific guidance for the specific situation a platform manufacturer is implementing or if performing the measurements specified in this section would not fully measure IPL Code and its corresponding configuration data into PCR[4] and PCR[5] respectively.

When BIOS boots a hard disk device or a hard disk-like device (e.g., a bootable USB device) with a primary boot sector, where the primary boot sector is typically an MBR, or has been otherwise identified as having an MBR, the MBR is recorded into both PCR[4] and PCR[5]. PCR[4] receives the IPL Code measurement and PCR[5] receives a measurement of the boot partition table.

Note: This specification only supports the classic MBR size of 512 bytes, not MBR sizes of 1K. It also does not support the GUID partition scheme. If these or similar unspecified situations arise, the platform manufacturer should measure code into PCR[4] and configuration data into PCR[5].

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.

When booting a hard disk device or a hard disk-like device with a primary boot sector with a classic MBR:

1. Bytes [0 – 1B7h] of the boot sector MUST be measured into PCR[4] using the event EV_IPL. The event field SHOULD be the string "MBR". See Section 11.3.1 (Event Types).
2. Bytes[1B8h – 1FFh] of the boot sector MUST be measured into PCR[5] using the event EV_IPL_PARTITION_DATA. The BIOS SHOULD use an event field of a string to help someone familiar with the device understand the measurement or SHOULD use the string "MBR PARTITION_TABLE". See Section 11.3.1 (Event Types).

8.2.4.2 **BIOS-Aware IPL Devices (BAID)**

Start of informative comment:

These are devices such as floppy drives, hard drives, CD-ROM drives, etc.

End of informative comment.

The IPL Code of BAID devices MUST be measured into PCR[4] before transferring control to the IPL Code using the event EV_IPL:

1. For hard disks or hard disk-like devices, see Section 8.2.4.1 (Hard Disk Device or Hard Disk-Like Devices).
2. For floppy disk devices or floppy disk-like devices with a boot sector where the boot sector is typically a file system boot sector, bytes [0 – 1FFh] MUST be measured into PCR[4] using the event EV_IPL. The event field SHOULD be the string “FLOPPY DISK”. See Section 11.3.1 (Event Types).
3. For other devices, measure the IPL Code into PCR[4] using the event EV_IPL. The platform manufacturer SHOULD use an event field of a string to help someone familiar with the platform understand the measurement or SHOULD use the string “IPL”.

8.2.4.3 Legacy IPL Devices

Start of informative comment:

Some legacy IPL devices predating the BIOS Boot Specification hook the INT 19h handler, potentially causing the S-RTM measurement chain of IPL devices to be untrustworthy. Evaluators of the S-RTM should be wary of validating a platform that contains such a legacy device. Because the BIOS may not know a device has hooked the INT 19h handler, it cannot record a measurement indicating hooking has occurred. An evaluator of the platform boot measurements must recognize the device’s measurement to determine the subsequent platform measurements are not trustworthy.

Option ROMs will have already been measured in PCR[0] or PCR[2], but if they are used as a boot device, their code is measured again in PCR[4].

It is the Option ROM’s responsibility to measure any additional code loaded or configuration data used.

End of informative comment.

1. For a platform to be complaint with this specification, it MUST be compliant with the BIOS Boot Specification.
2. Legacy IPL devices that hook the INT 19h handler are not compliant with this specification.

8.2.4.4 Boot Connection Vector (BCV)

Start of informative comment:

These include devices such as PnP SCSI cards with drive and Non-PnP card w/ expansion header. These cards generally require two BIOS calls, one to return their capabilities and another to have them hook INT 13h. Generally, for BCV devices, a generic driver is incorporated into the BIOS to interact with the device and was previously measured in PCR[0].

For the case of the cards that have an associated local mass storage device (SCSI cards with a bootable hard drive), then the BIOS must measure the device IPL such as a hard drive per Section 8.2.4.1.

End of informative comment.

1. For hard disks or hard disk-like devices, see Section 8.2.4.1 (Hard Disk Device or Hard Disk-Like Devices).
2. For other devices:
    a. The BIOS MUST measure the IPL Code into PCR[4] using the event EV_IPL. The platform manufacturer SHOULD use an event field of a string to help someone familiar with the platform understand the measurement or SHOULD use the string “BCV IPL”. See Section 11.3.1 (Event Types).
    b. The BIOS MUST measure additional configuration data for the device in PCR[5] using the event EV_IPL_PARTITION_DATA. The BIOS SHOULD use an event field of a string to help

someone familiar with the device understand the measurement or SHOULD use the string “BCV IPL DATA”. See Section 11.3.1 (Event Types).

8.2.4.5 Bootstrap Entry Vector (BEV)

Start of informative comment:

A device (generally a network card for PXE boot) that uses a BEV for booting requires being called through INT 19h. The device needs to measure the initial IPL image obtained from the device or network in PCR[4] prior to jumping to this code.

The device’s Option ROM image will have previously been measured in PCR[0] and/or PCR[2] and its configuration data in PCR[1] and/or PCR[3]. The Option ROM image and data may have been moved during the Option ROM initialization and its current location and size may be unknown to the BIOS so it cannot be measured again in PCR[4].

End of informative comment.

1. The Option ROM code MUST measure IPL Code it loads in PCR[4] using the events EV_IPL or EV_COMPACT_HASH. See Section 11.3.1 (Event Types). The Option ROM manufacturer SHOULD use an event field of a string to help someone familiar with the device understand the measurement or SHOULD use the string “BEV IPL”. See Section 11.3.1 (Event Types).
2. The Option ROM code MUST measure additional configuration data it uses in PCR[5] using the event EV_IPL_PARTITION_DATA. The Option ROM manufacturer SHOULD use an event field of a string to help someone familiar with the device understand the measurement or SHOULD use the string “BEV IPL DATA”. See Section 11.3.1 (Event Types).
3. If the BEV returns to the BIOS via an RETF, the BIOS MUST treat this as if the return was via INT 18h.

8.2.4.6 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.

1. The BIOS MUST measure the boot portion of the PARTIES Partition into PCR[4] using the event EV_IPL. The event field SHOULD be the string “PARTIES IPL”. See Section 11.3.1 (Event Types).
2. The BIOS MUST measure the BEER area of the PARTIES partition into PCR[5] using the event EV_IPL_PARTITION_DATA, even if the configuration data was already measured in PCR[1] or PCR[3]. The BIOS SHOULD use an event field of a string to help someone familiar with the device understand the measurement or SHOULD use the string “BEER IPL DATA”. See Section 11.3.1 (Event Types).
3. The BIOS MUST measure the partition table of the booted PARTIES partition in PCR[5] using the event EV_IPL_PARTITION_DATA. The event field SHOULD be the string “PARTIES IPL DATA”. See Section 11.3.1 (Event Types).

8.2.4.7 El Torito

Start of informative comment:

When the BIOS boots to a CDROM device that supports the El Torito specification, it first loads the Booting Catalog. If there is more than 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 CDROM that contains the image into memory and turns control over to this code.

Prior to jumping to this code, the BIOS measures 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.

1. The BIOS MUST measure the entire contents of the boot catalog in PCR[5] using the event EV_IPL_PARTITION_DATA. The BIOS SHOULD use an event field of a string to help someone familiar with the device understand the measurement or SHOULD use the string “BOOT CATALOG”. See Section 11.3.1 (Event Types).
2. The BIOS MUST measure the initial IPL image of the boot code in PCR[4] using the event EV_IPL. The event field SHOULD be the string “EL TORITO IPL”. See Section 11.3.1 (Event Types).

8.3 Power States, Transitions, and TPM Initialization

Start of informative comment:

Power Management

This section describes which Host Platform and OS actions are expected within power states and during transitions between power states. Careful power management needs to occur for the TPM device so its state accurately reflects the measurements of the current platform state. When the system is in the working state, the TPM state needs to persist, thus the TPM will also be in the working state. Some transitions, like turning off the machine, are expected to cause the TPM to lose its state. Other transitions, like entering and resuming from sleep, should save and restore TPM state if the OS and platform cooperate correctly; otherwise, the TPM will enter failure mode.

End of informative comment.

1. If the TPM is enumerated, the Host Platform MUST:
    a. Implement the Advanced Configuration and Power Interface (ACPI) Specification, Revision 2.0, July 2000 or later.
    b. Provide whatever resources (e.g., power) are needed for the TPM to behave as though it is in the D0 state while:
        i. The TPM device is in any device power state except for D3.
        ii. The TPM is not in D3 and system is in any of the power states:
            1. Legacy
            2. S0
            3. S1 OR
            4. S2
    c. Provide whatever resources (e.g., power) are needed for the TPM to maintain its saved state when the TPM enters D3 or the Host Platform enters S3. (For example, if the TPM places its saved state in NV Storage, it may not need power when in the D3 state. However, if the TPM does not use NV Storage to maintain its state when in the D3 state, the Host Platform needs to provide power for the device when the TPM is in the D3 state.)
2. When the Host Platform is in the S3 state, it SHOULD prohibit all TPM functions. (If the TPM does receive commands during S3, they MAY invalidate the saved TPM state.)

8.3.1 General Host Platform and OS Power Requirements

Start of informative comment:

Because power management of the TPM is under the control of the TBB, an OS cannot place the TPM device in D3 or transition the TPM out of D3 directly; however, an OS may initiate a transition to S3 that may cause the TBB to place the TPM in D3. If the system resumes from S3, the TBB will transition the TPM out of D3 and if the TPM state was saved properly, the TPM state is restored by the S-CRTM in the S3 resume path.

By implementing the requirements below, an OS can use a process for transitioning the TPM into and out of D3 that preserves the TPM's state saved with the TPM_SaveState command and restores it.

This specification does not support a Legacy (non-ACPI) OS transitioning the TPM into and out of the equivalent of the D3 state by saving and restoring the TPM state.

End of informative comment.

1. The TPM device state power management MUST NOT change outside the control of the TBB.
2. The TBB MUST ensure TPM is only in the D0, D1, or D2 state when the platform is not under the control of the TBB. (Note: Per the TPM Interface Specification, D1 and D2 TPM behavior is identical to D0.)
3. The Post-BIOS operating system SHOULD support ACPI.
4. After issuing a TPM_SaveState command, the OS SHOULD NOT issue TPM commands before transitioning to S3 without issuing another TPM_SaveState command.

8.3.2 Power State Transitions

Start of informative comment:

Each section below describes the behavior and requirements for entering or exiting each system power state. The only transitions allowed are those defined in the ACPI specification.

In general, the power state transition requirements for the BIOS can be summarized as:

1. The S-CRTM prepares the TPM for use when booting or resuming from S3.
2. If the S-CRTM is modified, the BIOS needs to reboot instead of resuming from S2 or S3.
3. TPM_INIT is only issued by the S-CRTM during boot or during S3 resume.
4. During boot and resuming from S3, the BIOS issues a TPM_ContinueSelfTest command.

End of informative comment.

8.3.2.1 Off to Legacy or S0 (Working)

Start of informative comment:

This transition is from an Off state to the platform power state that supports Legacy (non-ACPI) operating systems or ACPI operating systems. This is a normal boot process that initializes the TPM and begins the Static Root of Trust for Measurement. The Host Platform may later transition either direction between the Legacy and the S0 state.

If the TPM is not enumerated or the platform owner has disabled measurements, the CRTM may issue the TPM_Startup command such that the TPM is deactivated.

Transitioning out of S4 is similar to transitioning out of S5 except the BIOS or OS loader restore a memory image from persistent storage. The pre-boot components are measured normally for the current platform state, including the components that restore the memory image. When resuming from hibernation, the OS should be prepared for S-RTM measurements to be different from the previous boot.

It is possible physical presence operations or changes to the BIOS configuration may have changed the TPM state (e.g., a TPM may have changed from deactivated to activated via a physical presence operation).

The BIOS is responsible for issuing the TPM_ContinueSelfTest command. The Platform Manufacturer will know the characteristics of the TPM's Self-Test behavior and may optimize the issuance of the TPM_ContinueSelfTest command to optimize boot performance. It is recommended the BIOS issues the command at a time during boot when the TPM is unused so the Self-Test actions complete when the TPM is not needed. If the TPM is designed to not immediately return after receiving the TPM_ContinueSelfTest, the BIOS may start the command and poll for the return result later, allowing the BIOS to perform other boot actions during the execution of the TPM_ContinueSelfTest command. TPM behavior varies when the TPM receives a command while the Self-Test associated with the TPM_ContinueSelfTest command is occurring. Platform Manufacturers should carefully consider behavior of devices sending commands when a Self-Test is occurring and in some situations it may be appropriate to have the BIOS TPM driver encapsulate any retry logic necessary to allow the Self-Test to complete and retry TPM commands afterward.

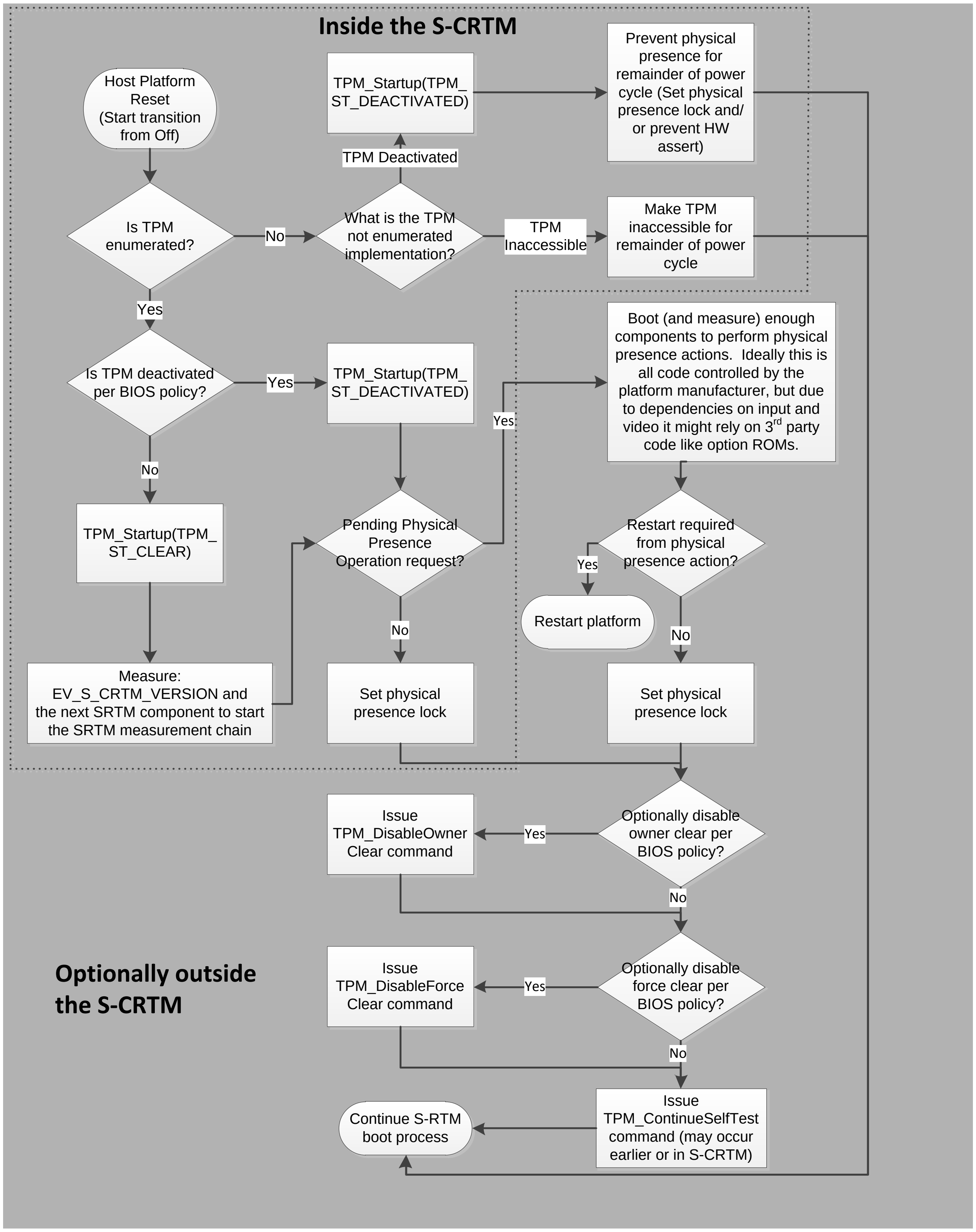
Inside the S-CRTM
Host Platform Reset (Start transition from Off)
Is TPM enumerated?
No
What is the TPM not enumerated implementation?
TPM Deactivated
TPM_Startup(TPM_ST_DEACTIVATED)
Prevent physical presence for remainder of power cycle (Set physical presence lock and/ or prevent HW assert)
TPM Inaccessible
Make TPM inaccessible for remainder of power cycle
Yes
Is TPM deactivated per BIOS policy?
Yes
TPM_Startup(TPM_ST_DEACTIVATED)
No
TPM_Startup(TPM_ST_CLEAR)
Measure: EV_S_CRTM_VERSION and the next SRTM component to start the SRTM measurement chain
Pending Physical Presence Operation request?
Yes
Boot (and measure) enough components to perform physical presence actions. Ideally this is all code controlled by the platform manufacturer, but due to dependencies on input and video it might rely on 3rd party code like option ROMs.
Restart required from physical presence action?
Yes
Restart platform
No
No
Set physical presence lock
Set physical presence lock
Optionally disable owner clear per BIOS policy?
Yes
Issue TPM_DisableOwnerClear command
No
Optionally outside the S-CRTM
Optionally disable force clear per BIOS policy?
Yes
Issue TPM_DisableForceClear command
No
Issue TPM_ContinueSelfTest command (may occur earlier or in S-CRTM)
Continue S-RTM boot process

Figure 7: BIOS actions during transitions from Off

Note: Additional requirements about the S-CRTM setting the physical presence lock are specified in Section 16.2: Indication of Physical Presence from the CRTM and are not duplicated here.

End of informative comment.

When transitioning from the Off state to the Legacy or S0 states:

1. The Host Platform MUST issue a TPM_INIT.
2. If the TPM interface is accessible, the S-CRTM MUST issue a TPM_Startup command with a startup type of TPM_ST_CLEAR or TPM_ST_DEACTIVATED.
3. If the TPM interface is accessible and the TPM is enumerated, the S-RTM MUST issue a TPM_ContinueSelfTest command prior to the first invocation of the first INT 19h call.
4. If the TPM is not deactivated and is enumerated, the S-CRTM MUST start the S-RTM measurement chain by recording integrity measurements during the boot process per Section 3.3.2 (Integrity Collection, Reporting, and Error Conditions).
5. If the TPM is not enumerated and the TPM interface is accessible:
    a. If the platform supports the Indication of Physical Presence from the S-CRTM, the platform manufacturer MUST set the physical presence lock (TPM_SF_PHYSICALPRESENCELOCK) during execution of the S-CRTM such that later software is unable to use physical presence to activate the TPM via the command TPM_PhysicalSetDeactivated.
    b. If the platform supports additional methods of asserting physical presence, the platform manufacturer MUST disable them.
6. IPL Code SHOULD be prepared for the TPM's Self-Test actions associated with the TPM_ContinueSelfTest command to be in progress when the IPL Code is launched.

8.3.2.2 Legacy or S0 (Working) to Off

Start of informative comment:

This transition is from the running OS to the Off state. In contrast to power loss, this is an orderly shutdown. Because TPM state is expected to be discarded after entering S5 (Off), there are no required actions.

The TCG Platform Reset Attack Mitigation Specification permits configuring the platform so memory is erased during boot under certain conditions. One variable is when the platform performed an orderly shutdown or unexpectedly lost power. Because this transition is an orderly shutdown, the OS should be able to purge secrets from the Host Platform memory if appropriate. Host Platforms implementing the TCG Platform Reset Attack Mitigation Specification may perform additional actions to avoid clearing memory on the next boot according to the TCG Platform Reset Attack Mitigation Specification.

End of informative comment.

8.3.2.3 S1 (Sleep) to S0 (Working), S0 to S1

Start of informative comment:

These transitions are between power states that all fully preserve the state of the TPM.

The TPM will not be in the D3 state when this transition occurs.

The S1 resume process does not jump to a resume vector. Instead, the processors continue execution where they stopped when entering S1.

If there are any changes to the Host Platform's components or configuration, measuring these changes is the responsibility of the OS.

End of informative comment.

The Host Platform MUST NOT issue a TPM_INIT.

8.3.2.4 S0 (Working) to S2 (Sleep)

Start of informative comment:

This transition is between power states that all fully preserve the state of the TPM.

The TPM will not be in the D3 state when this transition occurs.

The S2 resume process does jump to a reset vector because CPU context is lost while in S2.

A special case occurs when BIOS updates are installed and Host Platform code that executes on S2 or S3 resume is modified so the measurement in PCR[0] is no longer accurate. If a BIOS update has occurred since the last transition from S5 to S0 or to Legacy, the OS should reboot the Host Platform instead of allowing a transition to S2 or S3. If the OS does allow a transition to S2 or S3, as a failsafe, the S-CRTM is responsible for detecting modified BIOS code during the S2 or S3 resume and forcing a reboot or invalidating the contents of PCR[0]. The net result is upon a successful resume to the OS, the S-RTM measurements in PCRs[0-7] contain the correct measurements for the actual S2 or S3 resume code that executed or PCR[0] has been invalidated.

If there are any other changes to the Host Platform’s components or configuration, measuring these changes is the responsibility of the OS.

End of informative comment.

1. The Host Platform MUST NOT issue a TPM_INIT.
2. The OS SHOULD NOT transition from S0 to S2 if a BIOS update has occurred since the last transition from an Off state to S0 or to the Legacy state.

8.3.2.5 S2 (Sleep) to S0 (Working)

Start of informative comment:

This transition is between power states that fully preserve the state of the TPM.

The TPM will not be in the D3 state when this transition occurs.

Note: See the informative text in Section 8.3.2.4: S0 (Working) to S2.

This diagram illustrates the S-CRTM actions when resuming from S2:

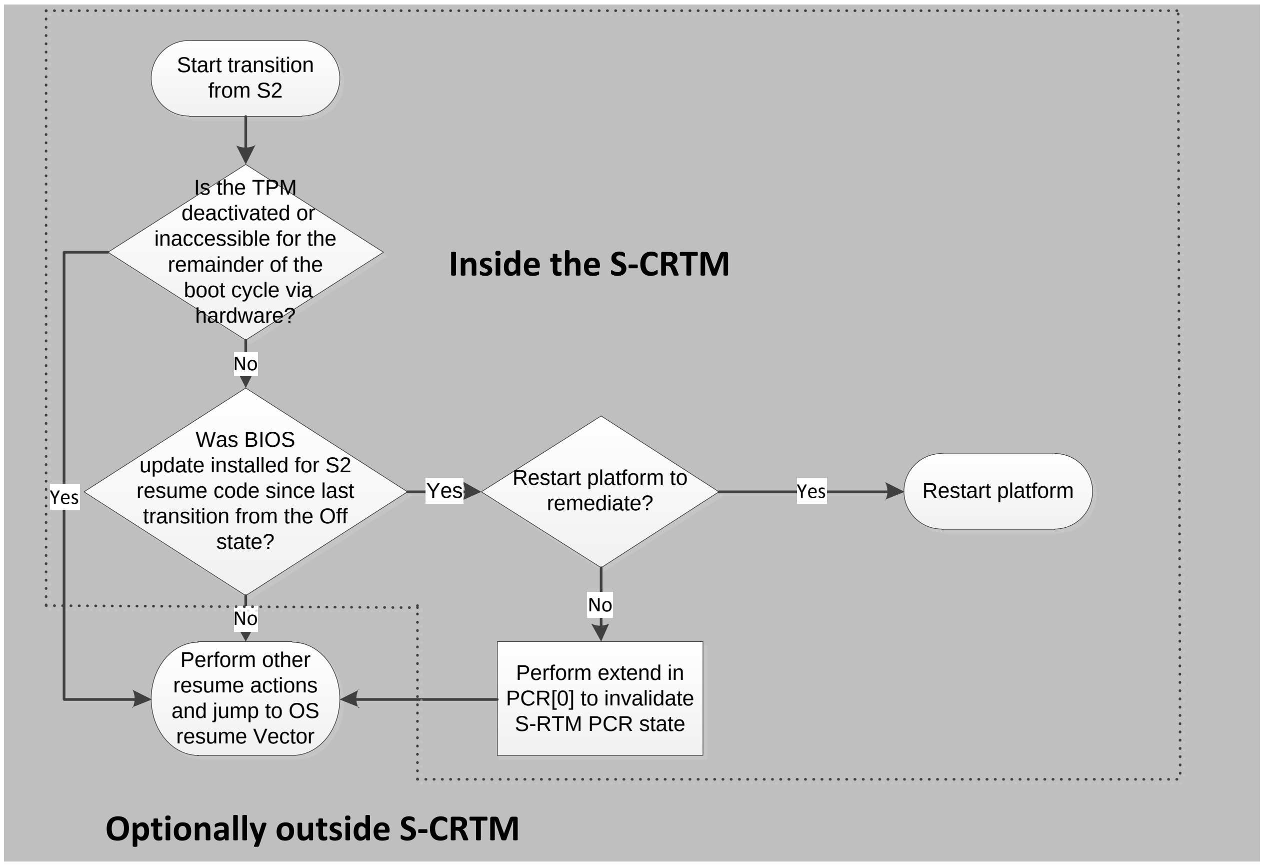
Start transition from S2
Is the TPM deactivated or inaccessible for the remainder of the boot cycle via hardware?
Inside the S-CRTM
No
Yes
Was BIOS update installed for S2 resume code since last transition from the Off state?
Yes
Restart platform to remediate?
Yes
Restart platform
No
No
Perform other resume actions and jump to OS resume Vector
Perform extend in PCR[0] to invalidate S-RTM PCR state
Optionally outside S-CRTM

Figure 8: BIOS actions for S2 resume

End of informative comment.

1. The Host Platform MUST NOT issue a TPM_INIT.
2. If the TPM is activated and enumerated:
   a. The S-CRTM MUST be able to determine whether there has been an update to any S2 resume portion of the BIOS since the previous transition from an Off state. Note: The CRTM SHOULD use a method that does not significantly add to the time it takes to resume from S2; in particular, it is not necessary for CRTM to re-measure BIOS code and compare this measurement to the measurement of BIOS code performed at the previous transition from S4 or S5. The determination of whether BIOS changed MAY be done using a simple flag.
   b. If the CRTM detects a modification to the S2 resume portion of the BIOS since the last transition from an Off state, the CRTM MUST either:
      i. Force the Host Platform to reboot, or
      ii. Make the contents of PCR[0] invalid by extending a value of 01h in PCR[0]. Note: No corresponding event is logged in the event log because the OS may be managing the log.

8.3.2.6 **S0 (Working) to S3 (Sleep)**

Start of informative comment:

This transition is from a working state to a sleep state that only needs to provide the necessary power to the TPM to maintain its saved state, not the full TPM context. Upon resume from S3, the saved TPM state is generally restored.

See the informative text in Section 8.3.2.4 (S0 (Working) to S2) for a special case regarding BIOS updates.

Entering into and exiting from the S3 state is a coordinated effort between the OS 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: D0 and 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 stating into the S0 power state. It is therefore a requirement that the Host Platform BIOS and OS participate in notifying the TPM of these transitions.

The OS driver notifies the TPM that it is about to transition to the D3 state and the system 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 makes the determination as to whether the Host Platform is resuming from an Off 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.

If TPM_Startup(TPM_ST_STATE) is called when there is no saved state to restore, the TPM enters failure mode, making it unavailable for the remainder of the power cycle. The only way to restore the TPM to a functional state is to perform a TPM_Init followed by a TPM_Startup, indicating to the TPM to clear its internal state (via transitioning from an Off state). If the TPM enters failure mode, the CRTM should still pass control to the OS. The TPM will only be able to perform limited actions until the platform boots again, but resuming the OS may be useful for the user.

Note that on systems without an OS TPM driver, on resume from S3, the TPM will always be in a failure mode because a TPM_SaveState command will not have been issued to the TPM prior to sleeping. Reboot is necessary before the TPM may be used.

The diagram below illustrates the logic for the S-CRTM actions when resuming from S3.

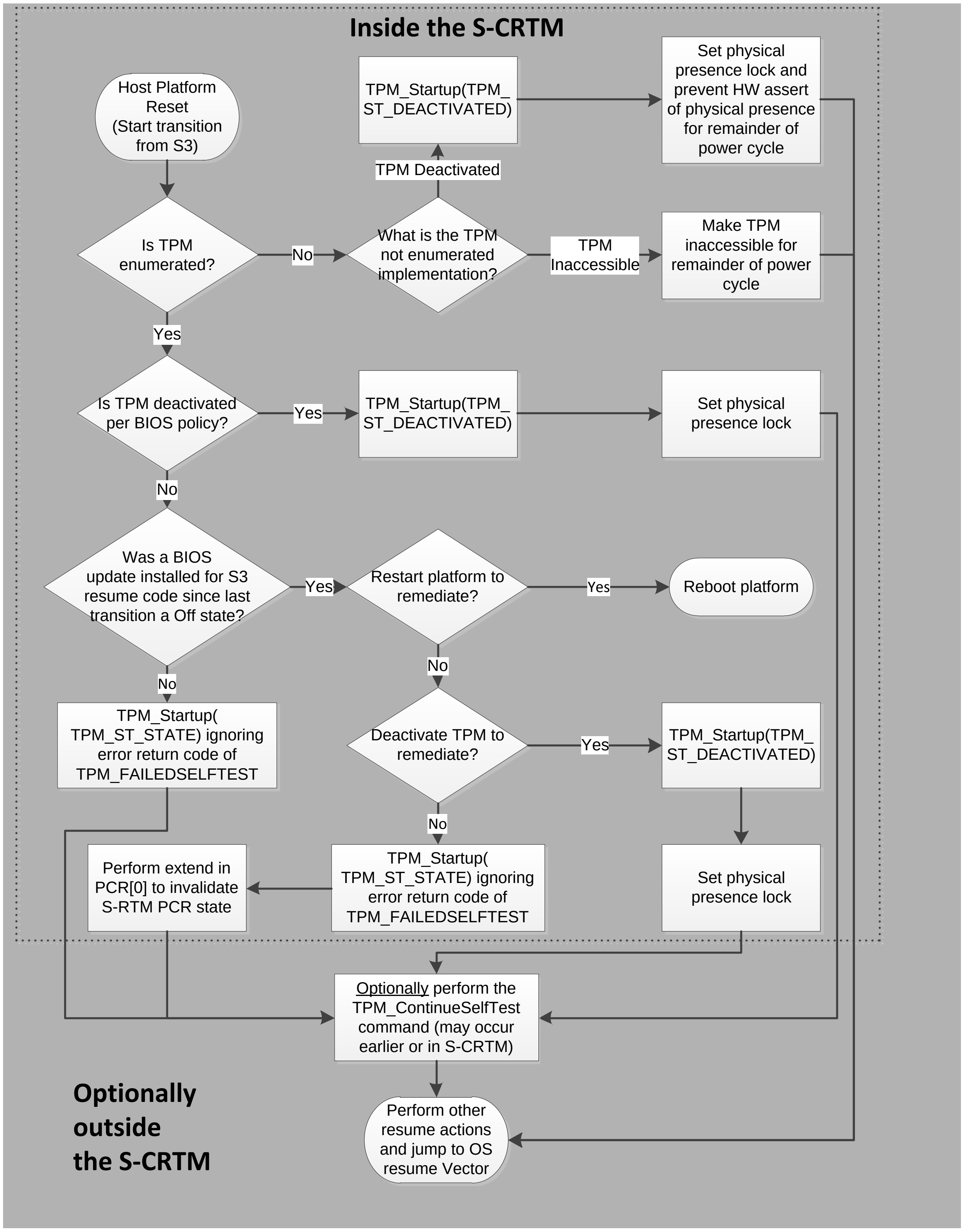
Inside the S-CRTM
Host Platform Reset (Start transition from S3)
Is TPM enumerated?
No
What is the TPM not enumerated implementation?
TPM Deactivated
TPM_Startup(TPM_ST_DEACTIVATED)
Set physical presence lock and prevent HW assert of physical presence for remainder of power cycle
TPM Inaccessible
Make TPM inaccessible for remainder of power cycle
Yes
Is TPM deactivated per BIOS policy?
Yes
TPM_Startup(TPM_ST_DEACTIVATED)
Set physical presence lock
No
Was a BIOS update installed for S3 resume code since last transition a Off state?
Yes
Restart platform to remediate?
Yes
Reboot platform
No
No
TPM_Startup(TPM_ST_STATE) ignoring error return code of TPM_FAILEDSELFTEST
Deactivate TPM to remediate?
Yes
TPM_Startup(TPM_ST_DEACTIVATED)
No
Perform extend in PCR[0] to invalidate S-RTM PCR state
TPM_Startup(TPM_ST_STATE) ignoring error return code of TPM_FAILEDSELFTEST
Set physical presence lock
Optionally perform the TPM_ContinueSelfTest command (may occur earlier or in S-CRTM)
Optionally outside the S-CRTM
Perform other resume actions and jump to OS resume Vector

Figure 9: BIOS actions during S3 resume

Note: In Figure 9 (BIOS actions during S3 resume), if the TPM_Startup(TPM_ST_STATE) returns TPM_FAILEDSELFTEST, it isn't necessary to extend an error in PCR[0] to invalidate the S-RTM PCR state because the TPM is in failure mode.

Note: See the requirement in Section 8.3.1 (General Host Platform and OS Power Requirements) for OS actions when placing the TPM in D3.

End of informative comment.

1. The OS SHOULD issue a TPM_SaveState command before transitioning to S3.
2. The Host Platform MUST NOT issue a TPM_INIT.
3. The OS SHOULD NOT transition from S0 to S3 if a BIOS update has occurred since the last transition from an Off state to S0 or to the Legacy state.

8.3.2.7 S3 (Sleep) to S0 (Working)

Start of informative comment:

This transition is a resume from an S3 suspend state. Host Platform Reset and TPM_INIT are asserted. The CRTM issues the TPM_Startup command, loading the previously saved state, without re-measuring Pre-Boot components. The CRTM passes control to the OS. If there are any changes to the Host Platform's components or configuration, measuring these changes is the responsibility of the OS.

See Section 8.3.2.6 (S0 (Working) to S3 (Sleep)) for more informative text regarding this transition.

The BIOS resume code won't jump to the OS Resume Vector while a command is executing. If it did, the OS driver would be unsure if it should allow the command to complete or abort it.

End of informative comment.

1. The Host Platform MUST issue a TPM_INIT.
2. The S-CRTM MUST issue a TPM_Startup command.
   a. Under these conditions the S-CRTM MAY issue a TPM_Startup(TPM_ST_DEACTIVATED) command:
      i. The TPM is not enumerated,
      ii. The TPM is deactivated per BIOS policy, or
      iii. A BIOS update has occurred for the S2 or S3 portion of the BIOS since the last transition from S5.
   b. Otherwise, the S-CRTM MUST issue the TPM_Startup(TPM_ST_STATE) command.
3. If TPM_Startup(TPM_ST_DEACTIVATED) is executed and the TPM interface is accessible, the S-CRTM MUST set the physical presence lock.
4. If the TPM is not enumerated and the TPM interface is accessible:
   a. If the platform supports the Indication of Physical Presence from the CRTM, the platform manufacturer MUST set the physical presence lock (TPM_SF_PHYSICALPRESENCELOCK) during execution of the S-CRTM such that later software is unable to use physical presence to activate the TPM via the command TPM_PhysicalSetDeactivated.
   b. If the platform supports additional methods of asserting physical presence, the platform manufacturer MUST disable them.
5. When issuing the TPM_Startup(TPM_ST_STATE), the S-CRTM SHOULD ignore an error resulting from the TPM entering failure mode (TPM_FAILEDSELFTEST). Note: This will happen when the TPM's state was not saved before entering S3.

6. If the TPM is activated and enumerated:
    a. The S-CRTM MUST be able to determine whether there has been an update to any S3 resume portion of the BIOS since the previous transition from an Off state. Note: The S-CRTM SHOULD use a method that does not significantly add to the time it takes to resume from S3; in particular, it is not necessary for S-CRTM to re-measure BIOS code and compare this measurement to the measurement of BIOS code performed at the previous transition from an Off state. The determination of whether BIOS changed MAY be done using a simple flag.
    b. If the S-CRTM detects a modification to the S3 resume portion of the BIOS since the last transition from an Off state, the S-CRTM MUST either:
        i. Force the Host Platform to reboot, or
        ii. Issue a TPM_Startup(TPM_ST_DEACTIVATED) command to deactivate the TPM, or
        iii. Make the contents of PCR[0] invalid by extending a value of 01h in PCR[0]. Note: No corresponding event is logged in the event log because the OS may be managing the log.
7. The BIOS MAY issue a TPM_ContinueSelfTest command.

    If the TPM TPM_ContinueSelfTest command immediately returns success and the TPM performs the Self-Test actions associated with the command asynchronously, the BIOS MAY launch the OS Resume Vector while the TPM_ContinueSelfTest Self-Test actions are still in progress.

8.3.2.8 Legacy to S0 (Working) or S0 to Legacy

Start of informative comment:

This transition is between a Host Platform state that supports a Legacy (non-ACPI aware) OS or a Host Platform state that supports an ACPI aware OS. This transition needs to preserve the TPM state because the PCR measurements represent the current platform state.

The TPM will not be in the D3 state when this transition occurs.

End of informative comment.

When transitioning from the Legacy state to the S0 state or when transitioning from the S0 state to the Legacy state:

1. The Host Platform MUST NOT issue a TPM_INIT.

9. TPM Enumeration and Discoverability

9.1 Enumeration

Start of informative comment:

If the TPM is not enumerated, the platform firmware needs to ensure the TPM is not used in an unintended manner. This could be done by deactivating the TPM during each boot while the firmware asserts physical presence or by other platform manufacturer-specific mechanisms.

An example of an incorrect implementation for a TPM that was configured to be “not enumerated” would be for firmware to return a value from TCG_StatusCheck that indicates the platform does not support TCG BIOS calls but still permits access to the TPM through the TPM’s memory-mapped I/O range with the TPM in a state of activated. In this incorrect implementation, the static Root of Trust for Measurement chain would stop at the first component that called TCG_StatusCheck to determine whether it should record measurements for later components in the TPM. Malicious software that ran later could masquerade as the component that called TCG_StatusCheck by building a false chain of measurements in the TPM for later components, then leveraging the “false” PCR values to unseal keys. Therefore, an acceptable mechanism to prevent unintended use of the TPM while not enumerated should somehow either block access to the TPM hardware or deactivate the TPM and set the physical presence lock (TPM_SF_PHYSICALPRESENCELOCK).

Likewise, when the TPM is not enumerated, simply disabling the TPM while leaving it activated is also not satisfactory because software could interact with the TPM’s memory-mapped I/O range to enable the TPM using commands like TPM_OwnerSetDisable.

When the TPM is not enumerated, the platform firmware is also responsible for ensuring the TPM continues to be deactivated or access to the hardware is blocked when the platform enters sleep (S3) and resumes.

The TPM enumeration state across S3/Resume cycles needs to be constant for several reasons. One is that operating systems do not expect the addition and or removal of a TPM device. Another is that when a TPM is present, the operating system will use the TPM_SaveState command to save the TPM state entering S3 and expect the BIOS to restore the state using the TPM_Startup command. If the TPM is not enumerated, the operating system will not issue a TPM_SaveState command to the TPM while entering S3. If the TPM enumeration state is different when resuming from S3, these actions would result in the TPM being in failure mode. Finally, if the TPM were to become enumerated upon resuming from S3, it would not have the S-CRTM measurements from the boot process.

There are two defined mechanism for implementing physical presence described in Sections 16.1 (Physical Switch) and 16.2 (Indication of Physical Presence from the CRTM).

If a Host Platform sets the TPM as deactivated when not enumerated and if it implements the Indication of Physical Presence from the Core Root of Trust for Measurement (CRTM), it needs to set the physical presence lock to prevent the use of physical presence to activate the TPM. If a Host Platform supports other mechanisms for physical presence, it may need to block access to the TPM or block the other mechanisms for asserting physical presence in addition to deactivating the TPM when the TPM is not enumerated. Otherwise, TPM_PhysicalSetDeactivated may still be used to change the TPM permanent flags, causing the TPM to be activated upon the next Host Platform Reset.

The default state of TPM enumeration is up to the Platform Manufacturer. If a platform is shipped with the TPM not enumerated, per the normative requirements below, the TPM must also be shipped deactivated.

End of informative comment.

9.2 TPM Discoverability

If a platform has a discoverable TPM, the TCPA table MUST be listed in the RSDT table as described in Section 10.2 (ACPI Table Usage (the TCPA ACPI Table)).

If a platform does not have a discoverable TPM, the TCPA table MUST NOT be listed in the RSDT table as described in Section 10.2 (ACPI Table Usage (the TCPA ACPI Table)).

9.3 Defined TPM Enumeration States

The allowed states for TPM enumeration are enumerated or not enumerated.

9.4 TPM Enumeration Requirements

1. Platform Manufacturers MAY provide an implementation-specific mechanism to control TPM enumeration on a Host Platform.
2. If a TPM is not discoverable on a Host Platform, the TPM MUST NOT be enumerated.
3. If the TPM is enumerated:
    a. The presence of the TPM MUST be indicated by the TPM device object being present in the ACPI table per Section 10.1 (Device Object for TPM).
    b. The platform manufacturer MUST comply with all of the requirements in the TPM Interface Specification.
    c. Any platform manufacturer-provided mechanism to transition the TPM to not enumerated MUST perform a Host Platform Reset.
    d. The TPM MUST remain enumerated on the Host Platform upon resuming from sleep (S3). (The platform entering and resuming from sleep MUST not change whether the TPM is enumerated or not.)
    e. The firmware MUST indicate support for the application-level interface (INT 1A TCG Functions).
    f. The method TCG_StatusCheck MUST NOT return TCG_PC_TPM_NOT_PRESENT.
    g. If the platform implements the command method for physical presence, the platform MUST implement the _DSM ACPI control method object for physical presence described in the TCG Physical Presence Interface Specification Version 1.2.
4. If the TPM is not enumerated:
    a. The presence of the TPM device object MUST NOT be present in the ACPI table per Section 10.1 (Device Object for TPM) or MUST return _STA "Not Present".
    b. The TPM MUST NOT be usable by platform software. This MUST be done by implementing one or both of these actions:
        i. The TPM interface MUST be inaccessible. (e.g., a hardware latch); or
        ii. All of the following MUST be implemented:
            1. The TPM MUST be deactivated.
            2. If the platform supports the Indication of Physical Presence from the CRTM, the platform manufacturer MUST set the physical presence lock (TPM_SF_PHYSICALPRESENCELOCK) during execution of the S-CRTM such that later software is unable to use physical presence to activate the TPM via the command TPM_PhysicalSetDeactivated.
            3. If the platform supports additional methods of asserting physical presence, the platform manufacturer MUST disable them.
    c. Any platform manufacturer-provided mechanism to transition the TPM to enumerated MUST perform a Host Platform Reset.
    d. The TPM MUST remain not enumerated on the Host Platform upon resuming from sleep. (The platform entering and resuming from sleep MUST not change whether the TPM is enumerated or not.)

e. The firmware MAY or MAY NOT indicate support for the application level interface (INT 1A TCG Functions).

f. The method TCG_StatusCheck MAY or MAY NOT return TCG_PC_TPM_NOT_PRESENT.

10. ACPI

10.1 Device Object for TPM

Start of informative comment:

In order to facilitate device discovery and OS 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 bus the TPM is on. The TPM device also contains at a minimum, an _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,)
            IRQ(Level, ActiveLow, Shared) {3,5,7,9,10,11,13,15}
    })
}
```

If legacy IO ports or any other hardware resources are decoded by the TPM, they are declared here also. Additionally, an _CID may be included. Per the ACPI specification, an _CID may be a single ID, or may be 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 IO ports declared in this example device object.

```
Device (TPM) {
    Name (_HID, EISAID("IFX0101"))
    Name (_CID, EISAID("PNP0C31"))
    Name (_CRS, ResourceTemplate() {
        Memory32Fixed (ReadWrite, 0xFED40000, 0x5000,)
        IRQ(Level, ActiveLow, Shared) {3,5,7,9,10,11,13,15}
        IO (Decode16, 0x4700, 0x4700, 0x01, 0x0C) //IO runtime 4700h-470Ch
    })
}
```

Some operating systems require all device resources to be claimed in ACPI. When the TPM is not enumerated, the TPM device object will not be present in ACPI. However, if appropriate, the platform manufacturer may claim the resources used by the TPM through other ACPI descriptions of the platform.

End of informative comment.

10.1.1 TPM Enumerated

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

When present, the ACPI device object representing the TPM MUST claim all hardware resources consumed by the TPM. This includes any legacy IO ports and other hardware resources.

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

10.1.2 TPM not Enumerated

While the TPM device is not enumerated, the platform ACPI namespace MUST NOT contain an ACPI device object for the TPM or MUST return _STA “Not Present”.

10.2 ACPI Table Usage (the TCPA ACPI Table)

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 Pre-Boot state to Post-Boot state.

The firmware “pins” the memory associated with the Pre-Boot 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 an Off state. 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.

The presence of the TCPA ACPI table is the standard mechanism that this specification provides for an operating system to discover if a TPM may be enumerated on the Host Platform. Operating systems should not expect to be able to use a TPM if it is not enumerated in the ACPI device list.

End of informative comment.

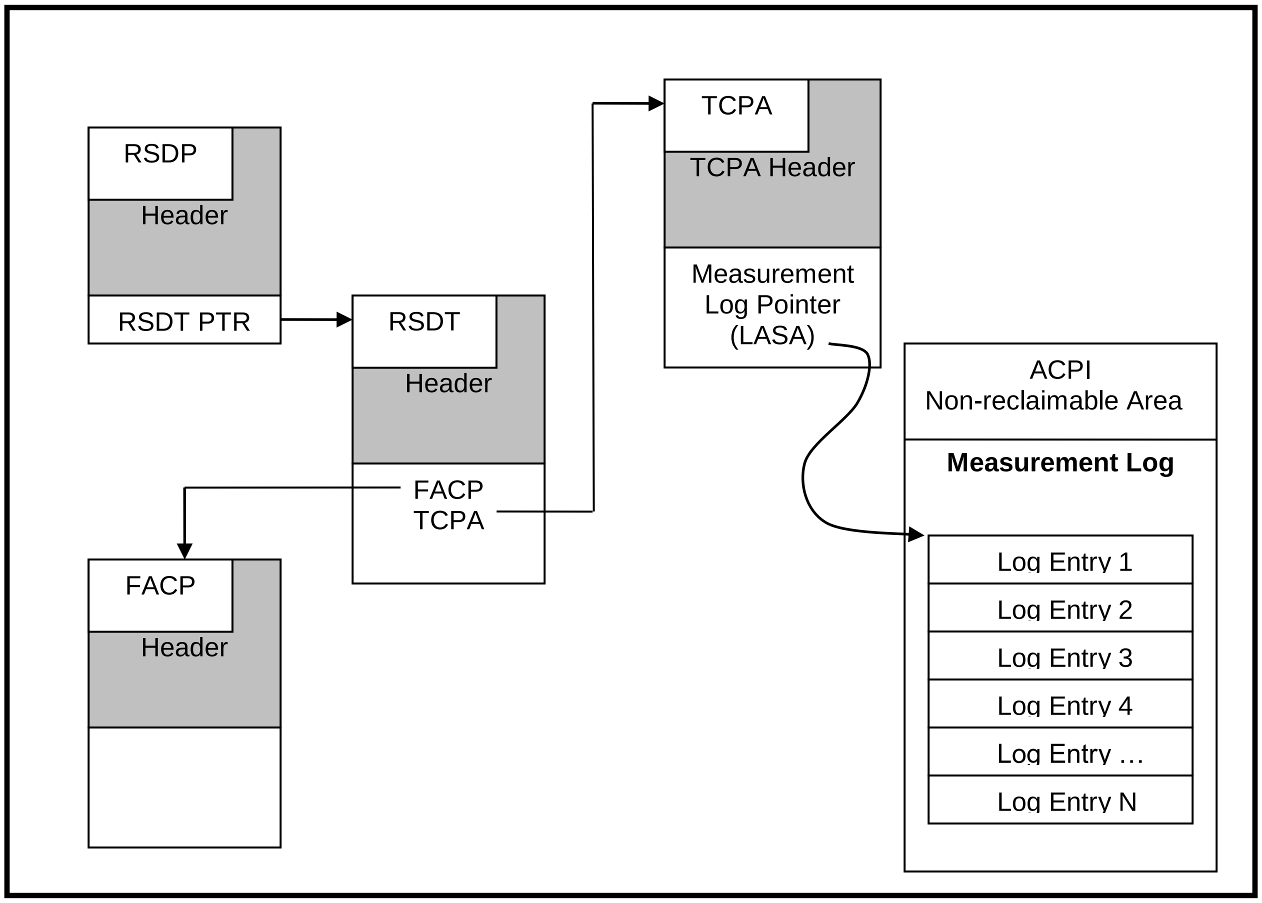

Figure 10: ACPI Table

1. The ACPI table indicated above as “TCPA” is defined in the TCG ACPI Specification.
2. The ACPI TCPA table MUST be listed in the RSDT ACPI table if the TPM “is discoverable” per the definition above in Section 1.2.11 (TPM Discoverability), regardless of whether the TPM is currently enumerated or not. The pointer to the TCPA table must be correct and the TCPA table MUST be populated.
3. The ACPI TCPA table MUST NOT be listed in the RSDT ACPI table if the TPM is “not discoverable” per the definition above in Section 1.2.11 (TPM Discoverability).
4. For the PC Client platform, when the TPM is enumerated, the Log Area Minimum Length (LAML) for the TCG event log MUST be at least 64 KB. Note: The TCG ACPI specification uses the field name “Log Area Minimum Length,” but the field value is the actual log area length reserved by the BIOS, not a lower bound.
5. When the TPM is not enumerated in the TCPA table, the minimum LAML for the TCG event log is zero.
6. All of the memory range of Log Area Start Address (LASA) through LASA + (LAML – 1) MUST be used exclusively to store event log event data. (For example, firmware storage of the location of the last event in the event log must be stored elsewhere.)
7. All of the memory range of LASA through LASA + (LAML – 1) MUST be initialized to zero by the BIOS.
8. The OS MAY use the buffer (LASA through LASA + LAML – 1) to store additional event data or other purposes after calling TCG_ShutdownPreBootInterface, even though the INT 1A TCG functions for adding additional log entries are not available.

11. Event Logging

11.1 Byte-ordering (Endianness)

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 that when software deals with the Host Platform itself (e.g., the PC Client data, structures, etc.), it uses 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. Changing within the context or purview would be inconsistent.

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.

11.1.1 TCG_PCClientPCREventStruct Structure

Start of informative comment:

For Version 1.1 of this specification, the corresponding TPM Main Specification defined the event log structure, more recent versions of this specification define the log. In Version 1.1, the event 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 OS-present software to retrieve the 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 10: ACPI Table is an instance of this TCG_PCClientPCREventStruc structure.

End of informative comment.

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

```
typedef struct tdTCG_PCClientPCREventStruc {
   UINT32                 pcrIndex;
   UINT32                 eventType;
   BYTE[20]               digest;
   UINT32                 eventDataSize;
   BYTE[eventDataSize]    event;
} TCG_PCClientPCREventStruc
```

Type	Name	Description
UINT32	pcrIndex	The PCR Index to which this event was extended.
UINT32	eventType	Defined in Section 11.3.1 Event Types.
BYTE[20]	digest	The value extended into the PCR identified by pcrIndex.
UINT32	eventDataSize	The size of the event.
BYTE[eventDataSize]	event	The data of the event.

Table 12: TCG PC Client PCR event structure

11.2 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.event described in Section 11.1.1 (TCG_PCClientPCREventStruct Structure) above.

The procedure for creating the measurement is to take a SHA-1 hash of the data contained within the TCG_PCClientPCREventStruc.event field, which in practice is the entire structure PlatformSpecificEventLogStruct (as defined in the TCG PC Specific Implementation Specification, Version 1.1, August 18, 2003). The resulting hash is placed into the field TCG_PCClientPCREventStruc.digest and used as the data in the TPM_Extend operation.

These structures are stored as an unstructured array within the ACPI data area as defined in Section 10.2 (ACPI Table Usage). None of the Pre-Boot 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 OS 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 measurement B and creation of a Measurement Log entry B, it is important to a verifier that the sequence of Measurement Log entry A and Measurement Log entry B be deterministic.

The event ordering within the event log is sequential. This convention provides a consistent view of when events occurred globally rather than relative to each PCR. For example, if the following sequence occurred, the event log would appear as (assuming the event just prior to event A was event #12):

Measure event A into PCR[2] ➔ Event 13

Measure event B into PCR[2] ➔ Event 14

Measure event C into PCR[3] ➔ Event 15

Measure event D into PCR[2] ➔ Event 16

And if someone parsed the events by PCRs, the events would appear as:

For PCR[2]:

Event 13, Event 14, Event 16

For PCR[3]:

Event 15

Note: Events are not numbered in the event log because the event structure does not contain an event number. Furthermore, from a security perspective, there is no way to detect if events in the log were re-ordered as long as the events for each individual PCR are sequential.

No log of events is recorded by the firmware if the TPM is deactivated. If the TPM is activated, a log is kept even if the TPM is disabled because the TPM may transition to enabled without a Host Platform Reset.

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 11.3.1 (Event Types).
    d. Fill A.event with either the data to be measured or description of data to be measured.
    e. A.digest = Depending on event type, either SHA-1{A.event} or the SHA-1{of data to be measured}.
    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 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. The first log entry in the measurement log MUST start at the Measurement Log Start Address. Subsequent measurements MUST be contiguous.
4. The first entry in the log MUST be the specification event. See Section 11.3.4.1 (Specification Event). Note: This event is not extended.
5. The sequence of the Measurement Log entries MUST be in the same sequence that the events were extended into the TPM PCRs.
6. If the TPM is deactivated, the BIOS MUST not record any log entries.

11.3 Event Descriptions

Start of informative comment:

Each event recorded in the Event Log is tagged as a particular event type. This section specifies all the event type tags that must or may be added to the Event Log on a PC client platform with a conventional BIOS.

End of informative comment.

11.3.1 Event Types

Start of informative comment:

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.

As of version 1.21 of this specification, the old event tags names from previous versions of this specification are omitted from Table 13: Event types. To research if an event type corresponds to an old event tag, please see a prior version of this specification.

End of informative comment.

The events in Table 13: Event types are defined for the field TCG_PCClientPCREventStruc.eventType.

Value	**Label**	**Description**
00h	EV_PREBOOT_CERT	The event type was partially specified prior to revision 1.21 and is considered deprecated and reserved for future use.
01h	EV_POST_CODE This event MUST extend the PCR	Used for PCR[0] only to record POST code, embedded SMM code, ACPI flash data, BIS code or manufacturer-controlled embedded option ROMs as a binary image. The digest field contains the SHA-1 hash of the code or data to be measured (e.g., POST portion of the BIOS). The event field SHOULD NOT contain the actual code or data, but MAY contain informative information about the POST code. For POST code, the event data SHOULD be “POST CODE”. For embedded SMM code, the event data SHOULD be “SMM CODE”. For ACPI flash data, the event data SHOULD be “ACPI DATA”. For BIS code, the event data SHOULD be “BIS CODE”. For embedded option ROMs, the event data SHOULD be “Embedded Option ROM”. See Section 3.3.3.1.
02h	EV_UNUSED	The event type was never used and is considered reserved.
03h	EV_NO_ACTION This event MUST NOT extend any PCR	The fields: pcrIndex and digest MUST contain the value 0. The event field contains informative data that was not extended into any PCR.
04h	EV_SEPARATOR This event MUST extend the PCRs 0 through 7 inclusive.	Used for PCRs[0, 1, 2, 3, 4, 5, 6 and 7]. Per Section 3.3.2.2, used to indicate an error occurred recording the CRTM, POST BIOS, or Embedded Option ROMs. For this situation, the digest MUST contain the value 01h. The event field is arbitrary to allow platform manufacturers to include an indication of what error occurred or other useful troubleshooting information. Per Section 8.2.3, used to delimit actions taken during the pre-OS and OS environments. For this situation, the digest value contains the SHA-1 hash of the event data. The event data size MUST be 4. The event field MUST contain the hex value 0xFFFFFFFF.
05h	EV_ACTION This event MUST extend the PCR	Used for PCRs [1, 2, 3, 4, 5, and 6]. The digest field contains the SHA-1 hash of the event field. A specific action measured as a string defined in Section 11.3.3 EV_ACTION Event Types.

Value	Label	Description
06h	EV_EVENT_TAG This event MUST extend the PCR	Used for PCRs [0, 1, 2, and 3]. The digest field contains the SHA-1 hash of the event field. The event field contains the structure defined in Section 11.3.2.1 (Tagged Event Log Structure). The structure contains the contents defined in Section 11.3.2.1. The digest field contains the SHA-1 hash of the event field.
07h	EV_S_CRTM_CONTENTS This event MUST extend the PCR	Used for PCR[0]. The digest field contains 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. See Section 3.3.3.1.
08h	EV_S_CRTM_VERSION This event MUST extend the PCR	Used for PCR[0] only. The digest field contains the SHA-1 hash of the event field. The event field contains the version string of the S-CRTM. See Section 3.3.3.1.
09h	EV_CPU_MICROCODE This event MUST extend the PCR	Used for PCR[1] only. The digest field contains the SHA-1 hash of the microcode patch applied. The event field contains a descriptor of the microcode patch. See Section 3.3.3.2.
0Ah	EV_PLATFORM_CONFIG_FLAGS This event MUST extend the PCR	Used in PCR[1] only. The digest field contains the SHA-1 hash of the event field. The event field contents are manufacturer implementation-specific but MUST represent whether each optional PCR[1] measurement is measured or not for the set of measurements that can be toggled by the platform owner. See Section 3.3.3.2.
0Bh	EV_TABLE_OF_DEVICES This event MUST extend the PCR	Used in PCR[1] only. The digest field contains the SHA-1 hash of the event field. The event field contents are manufacturer implementation-specific. 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. See Section 3.3.3.2.

Value	Label	Description
0Ch	EV_COMPACT_HASH This event MUST extend the PCR	May be used for any PCRs except 0, 1, 2, 3, or 6. The event field MAY be informative or MAY be hashed to generate the digest, depending on the component recording the event. This event is measured using the TCG_CompactHashLogExtendEvent. While it can be used by any component, it is typically used by IPL Code to measure events. The contents of the event field are specified by the caller. See Section 3.3.3.5.
0Dh	EV_IPL This event MUST extend the PCR	Used in PCR[4] and may be used in OS PCRs (e.g., 8-15). 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. For MBR code, the event data SHOULD be "MBR". For floppy disks, the event data SHOULD be "FLOPPY DISK". For BCV devices, the event data SHOULD be "BCV IPL". For BEV devices, the event data SHOULD be "BEV IPL". For a PARTIES partition, the event data SHOULD be "PARTIES IPL". For El Torito complaint devices, the event data SHOULD be "EL TORITO IPL". For other devices, the event data SHOULD be "IPL". See Section 3.3.3.5.

Value	Label	Description
0Eh	EV_IPL_PARTITION_DATA	Used for PCR[5] only. The data and/or the partition portion of the IPL Image. The digest contains the SHA-1 hash of the partition data. The event field SHOULD contain one of the informative comments below, or MAY contain a copy of the actual IPL data. For MBR code, the event data SHOULD be "MBR PARTITION_TABLE". For BCV devices, the event data SHOULD be "BCV IPL DATA". For BEV devices, the event data SHOULD be "BEV IPL DATA". For the BEER area of a PARTIES partition, the event data SHOULD be "BEER IPL DATA". For a PARTIES partition, the event data SHOULD be "PARTIES IPL DATA". For El Torito-compliant devices, the event data SHOULD be "BOOT CATALOG". See Section 3.3.3.6.
0Fh	EV_NONHOST_CODE This event MUST extend the PCR	Used for PCR[0] only. The executable component of any Non-Host Platform. The digest field contains the SHA-1 hash of the Non-Host Platform code. The event field may be determined by the platform manufacturer. See Section 3.3.3.1.
10h	EV_NONHOST_CONFIG This event MUST extend the PCR	Used for PCR[1] only. Configuration information or data associated with a Non-Host Platform. The digest field contains the SHA-1 hash of the Non-Host Platform configuration information. The event field is defined by the platform manufacturer. See Section 3.3.3.2.
11h	EV_NONHOST_INFO This event MUST extend the PCR	Used for PCR[0] only. Information about the presence of a Non-Host Platform. The digest field contains the SHA-1 hash of the event field. The event field could be, but is not required to be, information such as the Non-Host Platform manufacturer, model, type, version, etc. The event field is defined by the platform manufacturer. See Section 3.3.3.1.

Value	Label	Description
12h	EV_OMIT_BOOT_DEVICE_EVENTS	Used for PCR[4] only. The digest field contains the SHA-1 hash of the event field. The event field MUST be "BOOT ATTEMPTS OMITTED" in all caps. See Section 3.3.3.5.

Table 13: Event types

Start of informative comment:

The Event Type EV_PREBOOT_CERT has been deprecated in Version 1.21 of this specification along with Section 7.3. The deprecated event type information was:

Label	Value	Description
EV_PREBOOT_CERT This event MUST extend the PCR	00h	Used for PCR[2] only. The digest field contains the SHA-1 hash of the event field. The event field contains certificates such as the Validation Certificates.

Table 14: Deprecated event types

End of informative comment.

11.3.2 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.

11.3.2.1 Tagged Event Log Structure

The events shall be the following structure.

```
typedef struct tdTCG_PCClientTaggedEventStruct {
   UINT32              EventID;
   UINT32              EventDataSize;
   BYTE[EventDataSize]  EventData;
} TCG_PCClientTaggedEventStruct;
```

Type	Name	Description
UINT32	EventID	Tag as defined in the subsections below
UINT32	EventDataSize	Size of EventData
BYTE[EventDataSize]	EventData	EventData itself

Table 15: PC Client tagged event structure

11.3.2.2 Special Purpose structures

11.3.2.2.1 OptionROMExecuteStructure

```
typedef struct tdOptionROMExecuteStructure   {
   UINT16               Reserved;
   UINT16               PFA;
   BYTE[20]             HashData;
} OptionROMExecuteStructure;
```

Type	Name	Description
UINT16	Reserved	Reserved, MUST be set to zero
UINT16	PFA	Adapter PFA
BYTE[20]	HashData	Hash digest of the visible Option ROM code

Table 16: Option ROM execute structure

11.3.2.2.2 OptionROMConfigStructure

```
typedef struct tdOptionROMConfigStructure {
   UINT16               Reserved;
   UINT16               PFA;
   BYTE[]               OptionROMStruct;
} OptionROMConfigStruct;
```

Type	Name	Description
UINT16	Reserved	Reserved, MUST be set to zero
UINT16	PFA	Adapter PFA
BYTE[]	OptionROMStruct	Defined by Device Vendor. Compute the size from the total event size when reading.

Table 17: Option ROM config structure

11.3.2.3 Host Platform-Specific Event Tags

As stated in Section 11.1 (Byte-ordering (Endianness)), data is represented in Little Endian format.

The references to EventID are TCG_PCClientTaggedEventStruct.EventID.

The references to EventData are TCG_PCClientTaggedEventStruct.EventData.

11.3.2.3.1 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 structures

This structure is used to record measurements extended in PCR[1]. See Section 0.

11.3.2.3.2 BIS Certificate

The BIS Certificate SHALL be logged using the following:

EventID = 0002h

EventData[] = Raw BIS Certificate

This structure is used to record measurements extended in PCR[1]. See Section 3.3.3.2.

11.3.2.3.3 POST BIOS ROM Strings

The BIOS ROM Strings SHALL be logged using the following:

EventID = 0003h

EventData[] = Hash of POST BIOS ROM Strings

This structure is used to record measurements extended in PCR[1]. See Section 3.3.3.2.

11.3.2.3.4 ESCD

The ESCD SHALL be logged using the following:

EventID = 0004h

EventData[] = Hash of ESCD Data

This structure is used to record measurements extended in PCR[1]. See Section 3.3.3.2.

11.3.2.3.5 CMOS

The CMOS SHALL be logged using the following:

EventID = 0005h

EventData[] = Raw CMOS Data

This structure is used to record measurements extended in PCR[1]. See Section 3.3.3.2.

11.3.2.3.6 NVRAM

The NVRAM SHALL be logged using the following:

EventID = 0006h

EventData[] = Raw NVRAM contents

This structure is used to record measurements extended in PCR[1]. See Section 3.3.3.2.

11.3.2.3.7 Option ROM Execute

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

EventID = 0007h

EventData[] = OptionROMExecuteStructure

This structure is used to record measurements extended in PCR[2]. See Section 3.3.3.3.

11.3.2.3.8 Option ROM Configuration

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

EventID = 0008h

EventData[] = OptionROMConfigStructure

11.3.2.3.9 Option ROM Microcode Update

Deprecated because Option ROM Microcode updates are not recorded.

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

EventID = 000Ah

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

11.3.2.3.10 S-CRTM Version String

Deprecated by Event Type: EV_S_CRTM_VERSION

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

EventID = 000Bh

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

11.3.2.3.11 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.

11.3.2.3.12 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.

11.3.2.3.13 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.

11.3.3 EV_ACTION Event Types

For each EV_ACTION event produced by the platform, the BIOS MUST create the event indicated below in the following actions strings. 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 = 05h (the value for the EV_ACTION event type from the table in Section 11.3.1)

TCG_PCClientPCREventStruc.digest = the SHA-1 hash of the Event[] field

TCG_PCClientPCREventStruc.Event[] = ASCII string of the following:

Action Index[4]	**String**	**Purpose and Comments**	**PCR**
0	"Calling INT 19h"	BIOS is calling INT 19h. If no additional strings are posted in log, that means the software that 'hooked' the INT 19 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. 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"	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"	BIOS is IPL/Booting a BCV Device. The value 's' is an 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"	BIOS is IPL/Booting a BEV Device. The value 's' is an ASCII string supplied by the BEV device.	4

[4] **This is only used for reference within this document**

5	"Entering ROM Based Setup"	BIOS is entering ROM-Based Setup or Option ROM-Based Setup during pre-boot environment. If the Host Platform is designed to always perform a Host Platform Reset upon exit from the ROM-Based Setup Utility, then this measurement does not have to be made. See sections 3.3.3.2 (PCR[1] – Host Platform Configuration) and 3.3.3.4 (PCR[3] – Option ROM Configuration and Data).	1 or 3
6	"Booting to Parties n"	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"	User has entered the correct user password at the BIOS user interface. See note in Section 3.3.3.2 (PCR[1] – Host Platform Configuration).	1
8	"Administrator Password Entered"	User has entered the correct administrator password at the BIOS user interface. See note in Section 3.3.3.2 (PCR[1] – Host Platform Configuration).	1
9	"Password Failure"	The password typed at the BIOS user interface did not match the stored password after a specified number of retries.	1
10	"Wake Event n"	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 external SMBIOS Specification Section 3.3.2.2. Currently, only wake from S4 and S5 MAY be recorded.	6
11	"Boot Sequence User Intervention"	User altered the boot sequence.	1
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"	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 events 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

Table 18: EV_ACTION event types

11.3.4 EV_NO_ACTION Event Types

For each EV_NO_ACTION event produced by the platform, the BIOS MUST create the event indicated in this section. EV_NO_ACTION events MUST not extend the PCR, but they are recorded in the event log. They SHALL be logged using the following event structure:

TCG_PCClientPCREventStruc.pcrIndex = 0

TCG_PCClientPCREventStruc.eventType = 03h (the value for the EV_NO_ACTION event type from the table in Section 11.3.1)

TCG_PCClientPCREventStruc.digest = 0

TCG_PCClientPCREventStruc.eventDataSize = the size of TCG_PCClientPCREventStruc.event[] in bytes

TCG_PCClientPCREventStruc.event[] = one of the events described below

11.3.4.1 Specification Event

Start of informative comment:

This event provides identifying information about the version of this specification the BIOS supports. Note: The structure and its associated event were added in this specification in Version 1.21 (meaning specification version 1.2 errata 1). Any BIOS designed prior to this version of the PC Client Specification will not contain this event.

End of informative comment.

The event shall be the following structure:

```
typedef struct tdTCG_PCClientSpecIdEventStruct {
   BYTE[16]            signature;
   UINT32              platformClass;
   UINT8               specVersionMinor;
   UINT8               specVersionMajor;
   UINT8               specErrata;
   UINT8                reserved;
   UINT8                vendorInfoSize;
   BYTE[VendorInfoSize] vendorInfo;
} TCG_PCClientSpecIDEventStruct;
```

Type	Name	Description
BYTE[16]	signature	The null terminated ASCII string "Spec ID Event00". Must be set to {0x53, 0x70, 0x65, 0x63, 0x20, 0x49, 0x44, 0x20, 0x45, 0x76, 0x65, 0x6e, 0x74, 0x30, 0x30, 0x00}.
UINT32	platformClass	The value for the Platform Class. The enumeration is defined in the TCG ACPI Specification Client Common Header.
UINT8	specVersionMinor	The PC Client Specification minor version number this BIOS supports. Any BIOS supporting this version (1.21) MUST set this value to 02h. This value is from TCG_StatusCheck ECX bits 7-0.
UINT8	specVersionMajor	The PC Client Specification major version number this BIOS supports. Any BIOS supporting this version (1.21) MUST set this value to 01h. This value is from TCG_StatusCheck ECX bits 15-8.
UINT8	specErrata	The PC Client Specification errata for this specification this BIOS supports. Any BIOS supporting this version and errata (1.21) MUST set this value to 01h. This value is from TCG_StatusCheck ECX bits 23-16.
UINT8	reserved	Reserved and MUST be 0.
UINT8	vendorInfoSize	Size in bytes of the VendorInfo field. Maximum value MUST be FFh bytes.
BYTE[vendorInfoSize]	vendorInfo	Provided for use by the BIOS implementer. The value might be used, for example, to provide more detailed information about the specific BIOS such as BIOS revision numbers, etc. The values within this field are not standardized and are implementer-specific. Platform-specific or -unique information MUST NOT be provided in this field.

Table 19: Specification Identifier event structure

This structure is used to record a measurement for PCR[0] in the log, but it does not extend the PCR. See Section 3.3.3.1.

11.3.4.2 Vendor Specific EV_NO_ACTION Events

Start of informative comment:

Specific vendors may define their own events using this event type. The events will be present in the event log, but won't extend a PCR.

End of informative comment.

12. Implementation Overview

Start of informative comment:

The TCG interfaces in the Pre-Boot environment are: TCG Application Interface, TPCA Pre-Boot Driver Interface, and a set of ACPI structure definitions containing the event log. A diagram of these interfaces and their relationship is in Figure 11: Pre-Boot interfaces.

End of informative comment.

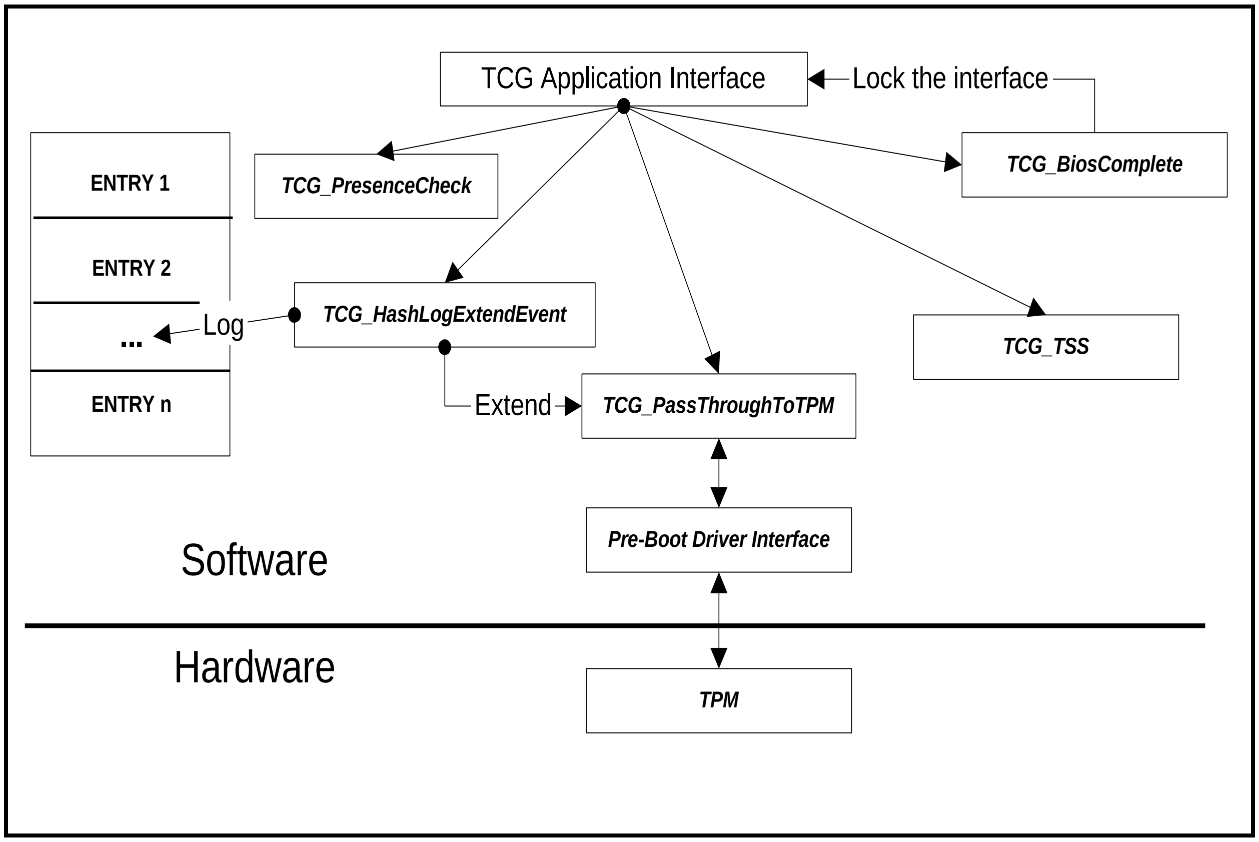

Figure 11: Pre-Boot interfaces

13. Application Level Interface – INT 1A TCG Functions

Start of informative comment:

This interface is intentionally lightweight, since the anticipated users of this interface are in the space-restricted Pre-Boot "space", 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.

Pre-Boot 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.

On entry:

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

On exit:

1. These INT 1A 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 64 KB 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.

13.1 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:

CF	=	0 if function completed successfully;
		1 if function failed
(EAX)	=	Return code
(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 non-TCG BIOS.

13.2 Indicators of Supported Functions

Start of informative comment:

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 1A Functions, there is a risk, though small, that a platform's BIOS implements the INT 1A 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.

There are two indicators that a function is supported.

The first method is use of the Carry Flag (CF). The Carry Flag is used by all INT 1A services as an indicator to the caller that the specific function within the INT 1A interface or function is implemented and successfully completed. If the carry flag is returned clear (value = 0), then the function is implemented and the return values are valid. If the carry flag is returned set (i.e., value = 1), then the function is either not implemented and the other return values are not valid or the function is implemented but the function failed.

The second method is to return the value 86h in EAX. This indicator should be used as secondary or in addition to the use of the Carry Flag.

End of informative comment.

1. All TCG INT 1A Functions, upon successful completion, MUST return with the Carry Flag (CF) clear (value = 0).
2. All TCG INT 1A Functions, upon failure, MUST return with the Carry Flag (CF) set (i.e., value = 1).

13.3 Return Codes

The following are the defined error codes the Pre-Boot functions MAY return. These, of course, apply only to INT 1A services that support the TCG APIs.

Return Code	**Value**	**Description**
TCG_PC_OK	0000h	The function returned successfully. For MPTPMTransmit function, see note within that function's definition.

Return Code	Value	Description
TCG_PC_TPMERROR	TCG_PC_OK + 01h | (TPM driver error << 16)	The TPM driver returned an error. This error is not generated by the INT 1A TCG function; rather, this error is a "pass-through" from the driver. It likely does indicate, depending on the function called, that the INT 1A TCG function failed. The upper 16 bits contain the actual error code returned by the driver as defined in Section 14 (Error and Return Codes).
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 TCG API function or a subfunction is not supported. This return code cannot be used as an indication that the INT 1A TCG functions are supported because if the platform's INT 1A service doesn't support the TCG functions, it will not know to return this error.
TCG_PC_INVALID_PARAM	TCG_PC_OK + 04h	An invalid parameter was passed. This applies only to the parameters passed to and parsed by the INT 1A TCG function, and does not indicate the status or return from the TPM. It is unlikely that any TPM operation was even performed as a result of the call to the INT 1A TCG function that returned this code.

Table 20: Pre-Boot function return codes

13.4 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.

Caller of INT 1A interface must allocate OP Buffer of at least 4 bytes long. (It can be 2 bytes since OPBLength field is defined as WORD, but the following Reserved WORD probably was abutted just to keep alignment.) Otherwise, the INT 1A interface behavior is undefined and may cause system crash.

On an error, the INT 1A 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. This includes the 4-byte entries that must be at the beginning of the output buffer.

13.5 Locality Issues

For each function specified below, the locality on exit must be in accordance with the locality selection as discussed in Section 3.2 (Locality State Requirements).

13.6 Self-Test Behavior

Start of informative comment:

The BIOS issues the TPM_ContinueSelfTest command during boot prior to the first invocation of INT 19h. Depending on the TPM's behavior after receiving the TPM_ContinueSelfTest command, the TPM may not be ready to receive additional commands immediately without returning TPM_DOING_SELFTEST or canceling the Self-Test actions and returning TPM_NEEDS_SELFTEST. Because issuing the TPM_ContinueSelfTest is a new requirement for this version of this specification, it is anticipated some Option ROMs or firmware code may not properly deal with a TPM_DOING_SELFTEST response from the TPM. Therefore, the INT 1Ah functions must insulate the callers of the INT 1Ah functions defined in Section 13 (Application Level Interface – INT 1A TCG Functions) from the TPM Self-Test behavior. It is expected Platform Manufacturers will understand the behavior of the TPM they select for the platform and will be able to implement the requirements of this section efficiently based on the TPM behavior. If a platform is designed to work with TPMs of varied behavior, the Platform Manufacturer may need to implement the requirements in this section conservatively.

Depending on the TPM behavior, per the TPM Main Specification, the TPM may execute the Self-Test actions asynchronously or synchronously. An asynchronous TPM_ContinueSelfTest implementation immediately returns success and then performs the actions associated with the Self-Test in the background. If additional commands are received while the actions are occurring, depending on the TPM implementation, the TPM may abort, pause, or complete the Self-Test actions. A synchronous TPM_ContinueSelfTest implementation performs all the Self-Test actions before the TPM signals completion of the TPM_ContinueSelfTest command.

If the TPM performs the Self-Test actions asynchronously, the BIOS implementation of the INT 1Ah functions defined in this section prevent the use of the INT 1Ah functions from inadvertently canceling the Self-Test actions. This is necessary because per the TPM Main Specification, some TPMs may cancel the Self-Test actions and return TPM_NEEDS_SELFTEST if a command is received while the Self-Test is running. A BIOS may need to delay sending commands to the TPM until enough time has elapsed for the TPM's Self-Test actions to have completed. Alternatively, a BIOS may send commands to the TPM, but trap the TPM_NEEDS_SELFTEST response and then reissue the TPM_ContinueSelfTest command and reissue the original TPM command.

If the TPM performs the Self-Test actions asynchronously and the TPM returns a TPM_NEEDS_SELFTEST response to commands it receives prior to completion of the Self Test actions, the implementation of the INT 1Ah functions in this section will trap the TPM_NEEDS_SELFTEST response from the TPM and resend the TPM command.

If the TPM performs the TPM_ContinueSelfTest command synchronously, the BIOS may start the TPM command executing and proceed with the boot process without waiting for the command to complete. In this situation, the firmware ensures the use of INT 1Ah functions defined in this section do not cancel the TPM_ContinueSelfTest command, but instead wait for it to complete before sending another command.

Some TPMs may permit a limited number of commands to occur while the Self-Test is in progress without disrupting the Self-Test actions. The firmware may permit these commands to be sent to the TPM while the Self-Test actions are still occurring to optimize boot performance.

In some scenarios, a TPM may not function correctly and could fail to complete the Self Test as expected by the Platform Manufacturer. In these error scenarios, the INT 1Ah functions defined in this section may return responses with TPM_NEEDS_SELFTEST or TPM_DOING_SELFTEST and attempt to perform the actions specified in Section 3.3.2.2 (Error Conditions).

End of informative comment.

After the BIOS issues the TPM_ContinueSelfTest command but before the TPM's Self-Test actions associated with the command are finished, for non-error scenarios, the BIOS implementation of the INT 1Ah functions in this section:

1. MUST prevent the cancellation of the TPM_ContinueSelfTest Self Test actions.
2. MUST prevent the return of a TPM_NEEDSSELFTEST result from a TPM response returned from the interface.

13.7 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 discoverable, this function MUST return the error: TCG_PC_TPM_NOT_PRESENT.
2. If the TPM is not enumerated, this function MAY return the error: TCG_PC_TPM_NOT_PRESENT.
3. If the TPM is discoverable but deactivated by TPM_Init or TPM_Startup, this function MUST return the error: TCG_PC_TPM_DEACTIVATED. (An exception is if the TPM is not enumerated and the method returns TCG_PC_TPM_NOT_PRESENT.) NOTE: 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	
(ECX)	=	Version and errata of the specification this BIOS supports.	
		Bits 7-0 (CL): 0x02	(TCG BIOS Minor Version (02h for version 1.21))
		Bits 15-8 (CH): 0x01	(TCG BIOS Major Version (01h for version 1.21))

		Bits 23-16: 0x01	(Sub-minor version or errata level (01h version 1.21))
		Bits 31-24: 0x00	(Reserved, must be 0)
(EDX)	=	BIOS TCG Feature Flags (None currently defined. MUST be set to 0)	
(ESI)	=	Absolute pointer to the beginning of the event log.	

If the TPM is not deactivated, this value must equal the “Measurement Log Pointer” value in the TCPA table. If the TPM is deactivated, the value is undefined.

(EDI) = Three conditions need to be accommodated:

1. If the event log is empty (i.e., no PCClientPCREventStruc structures), EDI MUST be set to zero (0).
2. If there is exactly one (1) PCClientPCREventStruc structure, EDI MUST be set to ESI.
3. If there is more than one (1) 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.

13.8 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, and 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 paramenter 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 data pointed to by the HashDataPtr 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 13.3 (Return Codes).
(DS:SI)	=	Referenced buffer updated to provide return results.

All other registers are preserved.

13.8.1 TCG_HashLogExtendEvent Input Parameter Block

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

13.8.1.1 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

Offset	Size	Field Name	Description
			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.

Table 21: Hash Log Extend Event Input Parameter Block Format 1

13.8.1.2 TCG_HashLogExtendEvent Input Parameter Block Format 2

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

Table 22: Hash Log Extend Event Input Parameter Block Format 2

13.8.2 TCG_HashLogExtendEvent Output Parameter Block

Note that the output parameter block has only one format.

Offset	Size	Field Name	Description
00h	WORD	OPBLength	The length, in bytes, of the output parameter block. If hash algorithm is SHA-1 with a 20-byte output, this value will be set by the INT 1A 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 TPM Main Specification states this hash functions use SHA-1, the size of this field will be 20h bytes.

Table 23: Hash Log Extend Event Output Parameter Block

13.9 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).

Note: Callers to the TCG_PassThroughToTPM may not be aware of the TPM's output data size when calling this function. The data in the MP driver's MPTPMTransmit function is required to return successfully if the TPM completes a function even if the data buffer provided is too small to contain the returned data. Because many of the TPM's operations are stateful, failing a TPM function that has a cryptographic operation just because of a buffer size inconsistency does not make sense. For this reason, callers to the TCG_PassThroughToTPM function are advised to check the return data size using the actual TPM's output within the TPMOperandOut buffer itself.

The TPM in and out Operands are defined in the TPM 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 13.3 (Return Codes)
(DS:SI)	=	Referenced buffer updated to provide return results.

All other registers are preserved.

13.9.1 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 an error if OPBLength is less than 4 bytes.

Offset	Size	Field Name	Description
06h	WORD	Reserved	
08h	BYTE	TPMOperandIn	The TPM Operand parameter block to send to the TPM.

Table 24: Pass-through to TPM input parameter block

13.9.2 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 1A 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.

Table 25: Pass-through to TPM output parameter block

13.10 TCG_ShutdownPreBootInterface

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

The IPL Code issues this call once the OS has its runtime access to the TPM available (e.g., an OS TPM driver) and no longer needs the firmware functionality for the INT 1A TCG functions to send commands to the TPM. This call causes the system firmware to no longer perform most INT 1A TCG functions, but instead respond with an error code 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

Execution of this function MUST not change the contents of the TCG event log buffer (address range LASA through LASA + LAMA – 1). See Section 10.2 (ACPI Table Usage (the TCPA ACPI Table)) for more information.

Calling this function is optional.

On entry:

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

On return:

(EAX)	=	Return Code as defined in Section 13.3 (Return Codes)

All other registers are preserved.

13.11 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 13.8 (TCG_HashLogExtendEvent) 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 where 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 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 13.3 (Return Codes)
(DS:SI)	=	Referenced buffer updated to provide return results

All other registers are preserved.

13.11.1 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.

Table 26: Hash Log Event Input Parameter Block

13.11.2 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 1A 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.

Table 27: Hash Log Event Output Parameter Block

13.12 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 13.3 (Return Codes)
(DS:SI)	=	Referenced buffer updated to provide return results

All other registers are preserved.

13.12.1 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.

Offset	Size	Field Name	Description
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.

Table 28: Hash All Input Parameter Block

13.13 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 such 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 13.3 (Return Codes)
(DS:SI)	=	Referenced buffer updated to provide return results

All other registers are preserved.

13.13.1 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

Offset	Size	Field Name	Description
			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.

Table 29: TSS input parameter block

13.13.2 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.

Table 30: TSS output parameter block

13.14 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, and then places the resulting TCG_PCClientPCREventStruc into the event log. This is 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-OS 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 error: TCG_INVALID_INPUT_PARA

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 be 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 Return Codes 10.2
(EDX)	=	Event number of the event that was logged

All other registers are preserved.

13.14.1 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.

13.14.2 TCG_BIOSVendorReserved

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

Reserved for vendor specific functions.

On entry:

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

14. Error and Return Codes

These are the return codes used by the MA / MP interfaces (if implemented) and returned by the INT 1A interface using return code TCG_PC_TPMERROR.

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

Start of informative comment:

If the BIOS does not perform the TPM_Extend functions, the TPM will be left in its initialization state. If, then, the OS driver can successfully communicate with the TPM, the TPM's Pre-Boot PCRs will be vulnerable to spoofing. For this reason, the BIOS must take action to prevent communication to the TPM for the duration of the platform's boot cycle until the next reset.

End of informative comment.

If either the Memory Present (MP) or the Memory Absent (MA) driver fails to communicate with the TPM, the BIOS MUST do one of the following:

1. Disable the connection to the TPM for the current boot cycle;
2. Take action to prevent the Host Platform from loading the operating system;
3. Perform a Host Platform Reset; or
4. Force transfer control of the Host Platform to a manufacturer-approved environment.

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 startup.
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	Timeout 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 timeout on the interface.
TCG_CHAR_READ_TIMEOUT	TPM_RET_BASE + 12h	Bytes read timeout 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 too short. Note: this is not returned as a result of a short buffer for the MPTPMTransmit function nor for the TCG_ PassThroughToTPM function.
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.
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	

Error Code	Value	Description
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 shut down 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	Starting value for return codes reserved for use by TPM vendors.

Table 31: Error and return codes

15. TPM Driver Interfaces

Start of informative comment:

Version 1.1 TPM and 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 driver.

There may be some situations where maintaining a standardized separation of function is beneficial. 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, it 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.

15.1 Module Architectures

15.1.1 TPM-Supplied BIOS Drivers

Start of informative comment:

The TPM vendor may supply one or two BIOS drivers in addition to the normal OS 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 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 is tightly controlled by its respective owner. It is desirable, but likely not globally achievable, that all BIOS to TPM driver interfaces will involve a minimal amount of non-standardized cooperation between the two module developers.

End of informative comment.

15.1.2 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. The table in Section 15.1.5.2 (BIOS Driver Header) describes what the header of the BIOS drivers will look like and where the driver code should start.

15.1.3 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 Core Root of Trust for Measurement (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 initial values they start at after a 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 fatal driver error (either Memory Absent or Memory Present), a fatal TPM communication channel error, or a fatal TPM error, the channel to 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; issuing a command to a port that stops communication to the TPM until a Host Platform Reset, etc. It is generally expected that the BIOS will take these actions since the driver can return timeout or other communication errors but, again, the implementation is Host Platform-specific.

15.1.4 Locality Access

The BIOS MUST access the TPM using only Locality 0.

15.1.5 BIOS Driver Header

15.1.5.1 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 driver.

15.1.5.2 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 from 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. 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. 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. Use of an alternative address is permitted for platforms supporting TPM 1.1b. Platforms supporting TPM 1.2 MUST use FED40000h.
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; i.e., don't try Base Address, causing 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 in none 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 zero.
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.[5]
XXh			Entry point into driver.

Table 32: BIOS driver header format

15.1.6 Basic Assumptions for Both BIOS Drivers

15.1.6.1 CMOSTimer

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.

[5] 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 convenience to the reader.

15.1.6.2 PC Motherboard Initialization

All PC 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.

15.1.6.3 Basic requirements

The BIOS drivers MUST fulfill the following requirements:

1. Be completely self-contained since no BIOS services should be used;
2. Check the validity of all the input parameters;
3. Include block chaining for the transmission of large data blocks to and from the TPM device; and
4. Be responsible for adding and removing all TPM-vendor-specific protocol information to the TCG-Transfer-Data (TCG-Command).

15.2 Memory Absent (MA) Driver

15.2.1 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 PC Motherboards containing an Integrated BIOS to implement this driver; instead, they may choose to implement only the Memory Present (MP) driver described in Section 15.3 (Memory Present (MP) Driver).

The purpose of the MA driver is to hash and extend the first portion of BIOS code before jumping to that code in a resource-constrained environment. The hash and extend operation for the MA driver and TPM should be optimized for performance.

End of informative comment.

15.2.2 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.

15.2.3 MA Driver Segments

In the Memory Absent driver, the data segment SHALL be flat (i.e., 0-based with 4-gigabyte limit for DS and ES data segments). The following is a detailed list of segments and their requirements:

Register	Description
CS	Normal “real-mode” code segment with no “protected-mode” artifacts. Upon entry to the MA driver, this MUST be based at the

	beginning to the BIOS Driver Header.
DS	0-based with 4-gigabyte limit
ES	0-based with 4-gigabyte 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.

Table 33: MA driver segments

15.2.4 MA Driver Limitations

1. The MA driver MUST be able to function in an environment without these resources:
   a. DMA
   b. IRQ
   c. Physical memory
2. MA driver register usage table (general purpose and segment register):

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 15.2.5. 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	BIOS sets this value, driver preserves value.

		Out: Preserved	
GS	16	In: Set by BIOS Out: Preserved	BIOS sets this value, driver preserves value.
SS	16	In (Segment)	Segment of the pointer to argument packet. See Section 15.2.5. Set by the caller.

Table 34: MA driver register usage

3. All other registers MAY be used as working registers by the MA driver without preserving them.
4. If the driver uses any other registers, that usage MUST be documented.

15.2.5 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 return via RETF.
   HeaderPtr      DD  ?   ; [IN] Pointer to the BIOS Driver Header (Reference 15.1.2).
   FunctionNum    DB  ?   ; [IN] Function number identifying the function to perform.
MADriverArgPacketStruct    ENDS
```

15.2.6 Parameters and Structures

15.2.6.1 Parameter pbInBuf

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

Table 35: Parameter pbInBuf

15.2.6.2 Parameter dwInPCRLen

DWORD *dwInPCRLen*	
Description	The 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).

Table 36: Parameter dwInPCRLen

15.2.6.3 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 TPM Main Specification (see TPM_Startup section in TPM Main Specification).

Table 37: Parameter bMAInitTPMFctId

15.2.6.4 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 TPM Main Specification.

Table 38: Parameter bMAPhyPresenceTPMCmdId

15.2.7 MA Driver Function Interface

The function number is contained in the FunctionNum field of the MADriverArgPacketStruct structure (reference Section 15.2.5). 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.

15.2.7.1 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 0).
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**

Table 39: Function MAInitTPM interface

15.2.7.2 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 is 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, and 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 15.2.6.1). *ECX = dwInPCRLen* PCRIndex and Length of the input buffer data (see 0).
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**

Table 40: Function MAHashAllExtendTPM interface

15.2.7.3 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, and 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 0).
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**

Table 41: Function MAPhysicalPresenceTPM interface

15.3 Memory Present (MP) Driver

15.3.1 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.

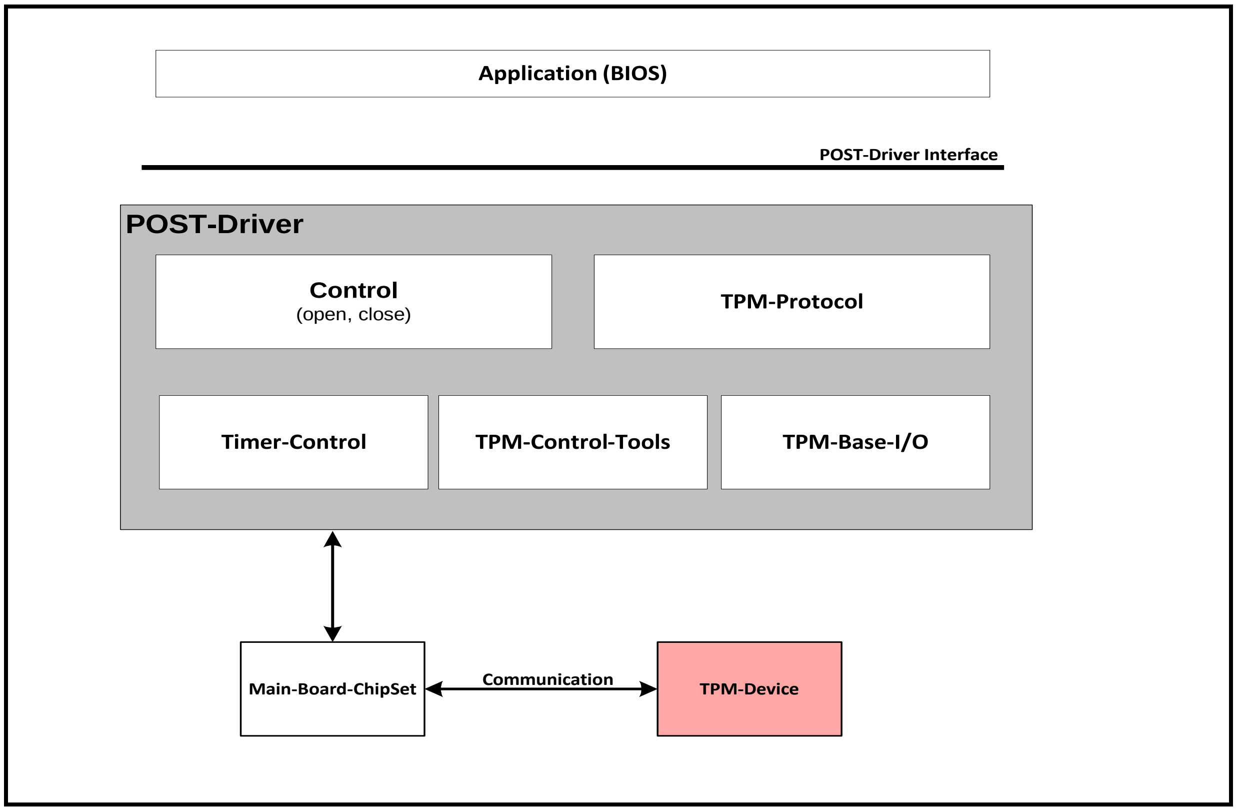

Figure 12: Pre-Boot driver interface

15.3.2 MP Driver Memory Model

In the Memory Present driver, the memory model MUST be a flat, 32-bit protected-mode. The code is position-independent 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.

15.3.3 MP Driver Segments

In the Memory Present driver, the data segment SHALL be a flat, 32-bit address. This means 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

Register	Description
CS	Normal 32-bit flat segment and MUST allow data access.
DS	0-based with 4-gigabyte limit.
ES	0-based with 4-gigabyte limit.
FS	The MP driver MUST NOT assume anything about this register. The MP driver MUST preserve the value and MUST not change limit.
GS	The MP driver MUST NOT assume anything

	about this register. The MP driver MUST preserve the value and MUST not change limit.
SS	The stack is a 32-bit stack. The driver MUST document the required amount of stack required.

Table 42: MP Driver Segments

15.3.4 MP Driver Limitations

1. No Interrupts are allowed. The MP driver MUST poll the TPM.
2. The MP driver MAY be relocated after MPInitTPM and at any time between call MP driver functions.
3. 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 MPInitTPM function MUST be recallable.
5. All registers not used for return parameters MUST be preserved.
6. MP driver needs to be built such that any data memory it requires is part of the body of the driver image.
7. If there is any paging, it MUST be identity mapped; i.e., virtual equals physical.
8. Any Host Processor feature that restricts execution by page MUST be disabled.

15.3.5 Parameters and Structures

15.3.5.1 Parameter pbInBuf

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

Table 43: Parameter pbInBuf

15.3.5.2 Parameter pbOutBuf

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

Table 44: Parameter pbOutBuf

15.3.5.3 Parameter dwInLen

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

Table 45: Parameter dwInLen

15.3.5.4 Parameter dwOutLen

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

Table 46: Parameter dwOutLen

15.3.5.5 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 also can be helpful in memoryless 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 output parameter:

As input (entry point of this function), it specifies the maximum number of bytes that 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.

15.3.5.6 Parameter lpTPMTransInfo

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

Table 47: Parameter lpTPMTransInfo

15.3.6 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 code in AL-Register.

15.3.6.1 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 also be 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 15.1.2).
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**

Table 48: Function MPInitTPM interface

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

This function 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 15.1.2).
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**

Table 49: Function MPCloseTPM interface

15.3.6.3 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 15.1.2).
Output Parameters	*None*
Return Value	*EAX = return value of this function* For the coding of the return value, see 15.3.7 (Return Values for MPGetTPMStatusInfo (Function: 03h)).

Table 50: Function MPGetTPMStatusInfo interface

15.3.6.4 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 deblocking 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 is added and removed by this function. The input and output buffer contains only TCG-Command-Param-Lists. The data streams are opaque to this function. This means that the TCG-Command-Param-Lists in these buffers will not be interpreted or reorganized by this function.

This function MUST fail and return a corresponding *errorCode* if:

1. No open connection to a TPM device is available;
2. The TPM returns no response;
3. The function-calling parameters are invalid (with the exception of pdwOutLen being too short as described below); or
4. The transmission of the data block to the TPM ends unsuccessfully.

If the pdwOutLen does not match the actual returned data size from the TPM, the MP driver MUST NOT treat this as an error. Provided the TPM function succeeds, the MP driver MUST copy the output data from the TPM into the bOutBuf location, truncating the data that will not fit per pdwOutLen.

BYTE MPTPMTransmit (**MPTPMTransmitEntryStruct** *lpTPMTransInfo*)*;	
Input Parameters	*ESI = pointer to a* TPMTransmitEntryStruct (see 15.3.5.5). *pbInBuf* Pointer to the input buffer containing the data (TCG command string) for the TPM device (see 15.3.5.1). *dwInLen* Length of the input buffer data (see 15.3.5.3).
Input/Output Parameters	*pdwOutLen* Pointer to store the length info of the received data (see 15.3.5.4). 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 15.3.5.2).

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**

Table 51: Function MPTPMTransmit interface

15.3.7 Return Values for MPGetTPMStatusInfo (Function: 03h)

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

DWORD-Return-Value	
Bit	**Description**
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 startup of the TPM firmware.
3	Timeout occurred during send process of the request sequence to the TPM device.
4	Response timeout 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.

Table 52: Function MPGetTPMStatusInfo return values

16. 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 a state where it is not owned. 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 integrity of the CRTM.

End of informative comment.

1. The PC Motherboard MAY provide a mechanism that provides proof of a physically present operator to the Host Platform. There may be TCG specifications describing optional methods for managing an OS to BIOS physical presence interface. An example is the TCG Physical Presence Interface Specification. See the TCG's set of documents for these specifications.
2. The manufacturing process for a Host Platform MUST set the TPM physicalPresenceLifetimeLock to TRUE.

16.1 Physical Switch

When activated, a physical switch or jumper or momentary button 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. Example: A DIP switch connected to the PC Motherboard that is within the Host Platform case.

16.2 Indication of Physical Presence from the CRTM

Start of informative comment:

The Physical Presence Interface 1.2 Specification defines the Persistent BIOS TPM Management Flags that allow some physical presence commands to execute without a physically present user. See the specification for more information.

Per the TPM Main Specification, when TPM_Startup(TPM_ST_STATE) completes successfully, it restores the TPM state including the state of the PhysicalPresenceLock. Section 8.3.2.7 (S3 (Sleep) to S0 (Working)) has requirements for the S-CRTM setting the PhysicalPresenceLock if TPM_Startup(TPM_ST_STATE) does not complete successfully when resuming from S3.

End of informative comment.

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, USB device, pressing a button) or when the Persistent BIOS TPM Management Flags are configured to not require physically present user confirmation of an operation. The CRTM MUST perform one of the two following sequences based on the indication of physical presence:

1. Physical presence NOT indicated: Exit normally, processing the remaining portions of the Pre-Boot environment.

   In this option, prior to exiting the CRTM, 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.

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 flag to FALSE and PhysicalPresenceLock flag to TRUE.

17. References

1. The Trusted Computing Group: http://www.trustedcomputinggroup.org
2. LPC Specification: http://www.intel.com/design/chipsets/industry/lpc.htm
3. Plug and Play BIOS Specification: http://download.intel.com/support/motherboards/desktop/sb/pnpbiosspecificationv10a.pdf
4. Advanced Configuration and Power Interface: http://www.acpi.info/
5. BIOS Boot Specification: http://www.phoenix.com/resources/specs-bbs101.pdf
6. Boot Integrity Services Application Programming Interface: http://download.intel.com/design/archives/wfm/downloads/bisspec.pdf
7. System Management BIOS Reference Specification: http://www.dmtf.org/standards/published_documents
8. “El Torito” Bootable CD-ROM Format Specification: http://download.intel.com/support/motherboards/desktop/sb/specscdrom.pdf
9. Preboot Execution Environment (PXE) Specification: http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf
10. PARTIES (Protected Area Run Time Interface Extension Services): http://t13.org/Documents/UploadedDocuments/project/d1367r3-PARTIES.doc
11. PCI Firmware 3.x Specification: http://www.pcisig.com/specifications/conventional/pci_firmware/
12. Extended System Configuration Data Specification: http://www.singlix.com/trdos/escd.pdf
13. Trusted Configuration Space for PCI Express ECN: http://www.pcisig.com/specifications/pciexpress/specifications/