TRUSTED® COMPUTING GROUP

SPECIFICATION

# TCG PC Client Platform Firmware Profile Specification

Family “2.0”
Level 00
Revision 1.05
December 4, 2019

Contact: admin@trustedcomputinggroup.org

## Work in Progress

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

PUBLIC REVIEW

# DISCLAIMERS, NOTICES, AND LICENSE TERMS

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

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



Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

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

# CHANGE HISTORY

| REVISION | DATE | DESCRIPTION |
|---|---|---|
| 1.00 21 | March 30, 2016 | Initial Release |
| 1.03 51 | March 20, 2017 | Published Update |
| 1.04 | April 2, 2018 | Incorporated Errata, member feedback, typos, and added Section 8.3 and Annex A |
| 1.05.02 | October 4, 2019 | • Added SPDM typedefs and updated PCR 2 and 3 text to add measurements.<br>• Reved UEFI_PLATFORM_FIRMWARE_BLOB and UEFI_HANDOFF_TABLE_POINTERS structures and added new event types for them.<br>• Updated the SP800-155 event structure based on IWG feedback<br>• Updated section 5. |
| 1.05.03 | November 4, 2019 | • Addressed feedback |
| 1.05.04 | November 7, 2019 | • Fixed cross references |

# ACKNOWLEDGEMENTS

The writing of a specification, particularly a security specification, takes many hours for both development and review. The TCG would like to acknowledge the contribution of those individuals (listed below) and the companies who allowed them to volunteer their time to the development of this specification.

Special thanks are due to Amy Nelson and Rob Spiger, who served as Chairs of the PC Client Working Group during the development of this specification.

The TCG would also like to give special thanks to Amy Nelson who served as the editor of this specification.

# CONTRIBUTORS

# CORRECTIONS AND COMMENTS

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

# TPM DEPENDANCIES AND REQUIREMENTS

1. The TPM used for Host Platforms claiming adherence to this specification SHALL be compliant with the *TPM Library Specification; Family 2.0; Level 00; Revision 01.38* or later.
2. The TPM used for Host Platforms claiming adherence to this specification SHALL be compliant with the TCG PC Client Specific Platform TPM Profile for TPM 2.0 Version 1.00, Revision 1.00 or later.
3. The Platform Class for platforms claiming adherence to this specification SHALL be registered with the TCG administrator.
4. Host Platforms claiming adherence to this specification SHALL be compliant with the TCG ACPI Specification Family 1.2 and 2.0, Level 00, revision .37.



 

# CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# 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 Library Specification Family 2.0 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 PC Client firmware architectures. Specifically, this document defines:

1. Usage of PCR registers in the Pre-Operating System state through the transition to OS-Present state.
2. How Platform Firmware, or a component thereof, contributes to the Root of Trust for Measurement (RTM) of the platform, specifically through extending digests (measurement) of code to a TPM based Platform Configuration Register (PCR) and documentation of that measurement in a log (event log).
3. Behavior entering, during, and exiting power and initialization states.
4. Guidelines for Option ROMS.

This specification is based on the TPM Library Specification Family 2.0, Level 00 Revision 1.38. 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 Library Specification will not be defined in this document. If there is a conflict in interpretation between this and the TPM Library Specification, the concept or functional description as defined in the TPM Library 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 by a measurement log. While not trusted on its own, the measurement log contains a richer set of information than do the PCR contents. The measurement log is not intended to be used to validate the PCR contents, rather, the PCR contents are used to provide the validation of the measurement log. Additional trust in the measurement is gained from comparing the PCR contents to a Reference Integrity Manifest (RIM), as specified in the TCG Reference Integrity Manifest Information Model. A RIM, provided by the Platform Manufacturer or Platform Firmware provider, is a signed representation 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.

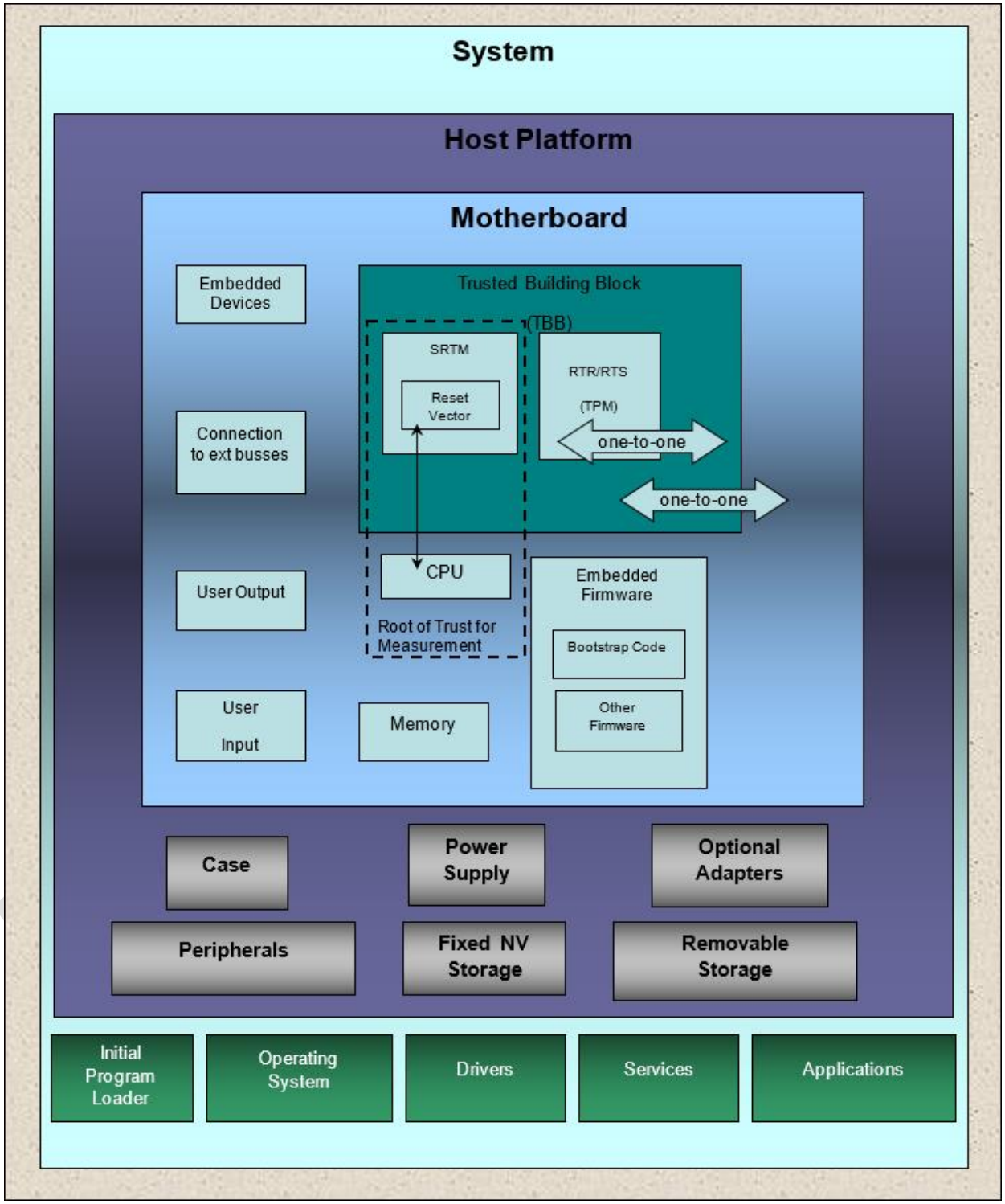


**Figure 1 PC Client Platform Architectural Diagram**

## 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 Library Specification identified in the TPM Dependency and Requirements section above.

### 1.2.2 Trusted Building Block (TBB)

A TBB consists of hardware and/or software that establishes a root of trust for measurement (provides an integrity measurement) and provides connectivity between an SRTM, the TPM, the PC Motherboard, the platform reset, and the TPM physical presence signal.

Because the SRTM 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 enabled 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 in 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 SRTM initiates the measurement of the state of the hardware and software environment in a platform. Three data components are involved in creation of 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.

A PC Client platform has a TBB consisting of the SRTM and the TPM. A PC Client has only one TPM, one SRTM, 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 SRTM 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 Platform Firmware, 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.

### 1.2.3 Roots of Trust

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

#### 1.2.3.1 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.

#### 1.2.3.2 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.

#### 1.2.3.3 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 SRTM and are measured as part of the SRTM. 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 Platform Firmware to detect the Operator by someone pressing a “function” key while the platform is under control of Platform Firmware, 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 SRTM measurements using events like the SRTM version identifier (EV_S_CRTM_VERSION) or a measurement like EV_POST_CODE.

**End of informative comment**

### 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

In the scope of this specification a Non-Host Platform is a self-contained execution environment within the system. These platforms execute in an environment separate from the Host Platform components. This is not to be confused with a peripheral, which, while potentially containing a powerful engine, only services and responds to requests from the Host Platform. For example, an 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 SRTM, 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 SRTM, is "effectively simultaneous" with the Host TPM's reset.

TPM Device Reset is defined in the PC Client Specific Platform TPM Profile for TPM 2.0, Section 1.1 (Terminology).

**End of informative comment**

#### 1.2.7.1 Reset 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 state 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 SRTM issues the TPM2_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 SRTM.
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 Platform Firmware 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, Platform Firmware MUST check the return status from the configuration utility and if the configuration utility requests a system reset, Platform Firmware 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 Control Surface

#### 1.2.10.1 TPM Visibility

**Start of informative comment**

Generally, OS loader components use UEFI Services to identify and interact with the TPM. A fully loaded OS uses the ACPI table entry (described in the *TCG ACPI Specification for TPM Family 1.2 and 2.0*) to identify the TPM and an OS resident TPM driver to interact with the TPM.

Platforms with a TPM intended for use by a fully loaded operating system populate the TPM device in the ACPI table of devices for the OS. Specifically, the TPM device will be represented by a device node in ACPI Source Language (ASL) code (see Section 8 ACPI) which will return a status (_STA()) indicating the TPM is present. Additionally, the TCG2_EFI_PROTOCOL will be present and the EFI_TCG2_BOOT_SERVICE_CAPABILITY.TPMPresentFlag will be TRUE (see the TCG EFI Protocol Specification revision 13 or later).

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 hide 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. An alternative use case for this functionality would be an end user with a dual boot Operating System environment where one Operating System makes full use of the TPM and the second OS does not support TPM. The requirements for hiding the TPM are contained in 8.1.1 TPM Visible.

If the TPM is visible to the Operating System environment, a platform manufacturer may provide an option to disable the TPM or one or more of the hierarchies of the TPM. In this case, the TPM would still be visible to the OS which would have the means, through the Physical Presence Interface, to re-enable the disabled portions of the TPM on a subsequent boot. Providing this capability could cause OS or application features tied to the TPM to stop working, so care should be taken in presentation of these options to end-users.

Mechanisms to control visibility of the TPM to an operating system are platform manufacturer-specific. An example is a BIOS configuration option to hide or show the TPM to the OS.

**End of informative comment**

A TPM is visible if the Platform Firmware indicates the presence of the TPM device to the OS. A TPM is hidden if the Platform Firmware does not indicate the presence of the TPM device to the OS. See Section 8.1.2 TPM Hidden but Discoverable for more information.

#### 1.2.10.2 TPM Discoverability

**Start of informative comment**

Because some platform manufacturers may provide a mechanism to prevent the discovery of a TPM by an Operating System which does not support it (thus leading to devices with no drivers loaded in the OS), an alternative mechanism is used to indicate whether a TPM is physically present on a platform. The Host Platform may be configured through some vendor proprietary mechanism to make the TPM visible to the OS and allow usage of 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 visible to the operating system or not. The existence of the TPM2 ACPI table means a mechanism exists on the platform for a platform owner to make the TPM visible 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 the TPM could be made visible to the OS by defined platform owner action. 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 enable the TPM. In the case that the TPM is not discoverable, the Platform Firmware disables the endorsement and storage hierarchies.

#### 1.2.10.3 Enabling TPM

**Start of informative comment**

The TPM 2.0 device starts up with all of its functionality fully enabled. The Platform Firmware does not need to take any action, other than sending the TPM2_Startup and TPM2_SelfTest commands to make the TPM device itself ready for use. However, the Platform Firmware does need to make the TPM visible unless action is required to make it not discoverable. In the case where an end user would like to turn off TPM functions or prevent an Operating System environment from using it for one boot cycle or more, the Platform Firmware may support a mechanism to disable the TPM.

**End of informative comment**

A TPM is enabled on a Host Platform if it is visible to the Operating System and all available hierarchies are enabled. A TPM is disabled on a Host Platform if it is visible to the Operating System and at least the Storage and Endorsement Hierarchies are disabled. A disabled TPM may have the Platform Hierarchy disabled.

### 1.2.11 Boot State Transition

The transition between Pre-OS and OS-Present states is the first invocation of Ready to Boot or equivalent.

### 1.2.12 Establishing the Chain of Trust

#### 1.2.12.1 Binding 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 Library Specification Family 2.0 Level 00 Revision 1.38, Part 1, Section 9.4.3.3 (RTR Binding to a Platform). A platform compliant with this specification SHALL ensure the TPM is powered (on/off) and reset at the same time as the host CPU.

#### 1.2.12.2 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**

 

### 1.2.13 Locality

**Start of informative comment**

**Hardware Proof of the Source of a Command**

The TPM supports several authorization types to provide a trusted path between a user and a TPM's object (e.g., a TPM key or NV area). Common authorization methods are password, secret key, and public key. These methods rely on shared or provisioned secrets. Some technologies require a trusted path between hardware components. Methods involving shared or provisioned secrets are not practical for these hardware components. Locality uses hardware addressing to enforce a trusted path between hardware components. By restricting the addresses at which some hardware components can address the TPM, locality provides proof of the source component's identity. This allows TPM object policies to apply to specific hardware components enforced by the hardware.

Note that the enforcement agent for the trusted path (i.e., the authorization of the source of the command) is not in the TPM, rather it is the hardware controlling the channel to the TPM. The TPM will take commands sent to it at any locality and act on those commands, giving that command access to that locality's objects.

The PC Client defined TPM supports 5 Localities: 0-4. These are divided into five 4K Memory-Mapped IO (MMIO) ranges starting at the base address of the TPM's lowest address range. Localities are implemented by assigning each Locality to one of the 4K MMIO ranges. With the PC Client TPM mapped to the FED4_XXXX range, Locality 0 is mapped to FED4_0XXX; Locality 1 to FED4_1XXX; through Locality 4 to FED4_4XXX.

Locality 0 is designated as the default Locality. This is usually the platform's boot firmware, OS and applications. Locality 4 is designated for use by one of the CRTM's (Static or dynamic). Locality 1-3 are used by higher privileged components or software.

Note: the above description applies to hardware localities. There are also software localities used, for example, in virtualization, which are not associated with hardware addresses. These software localities are outside the scope of the PC Client TPM discussed here.

**Sharing of the TPM**

The TPM is a single-threaded, non-preemptive device and can therefore service only one request at a time. With localities, however, it is possible to have multiple domains sending commands to the TPM. The PC Client Platform TPM Profile provides a method to prevent one domain retaining control of the TPM until it has completed the current command.

To access the TPM, a driver within a domain (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 but without a session handle). When completed, and especially when a different domain is requesting use of the TPM, the software should then release the TPM. This allows the TPM to be shared between the various processes within the platform.

**General Description of the CRTMs**

Locality provides an expression of a CRTM called the Dynamic CRTM (DRTM) that is independent of the Host Platform Reset. 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**

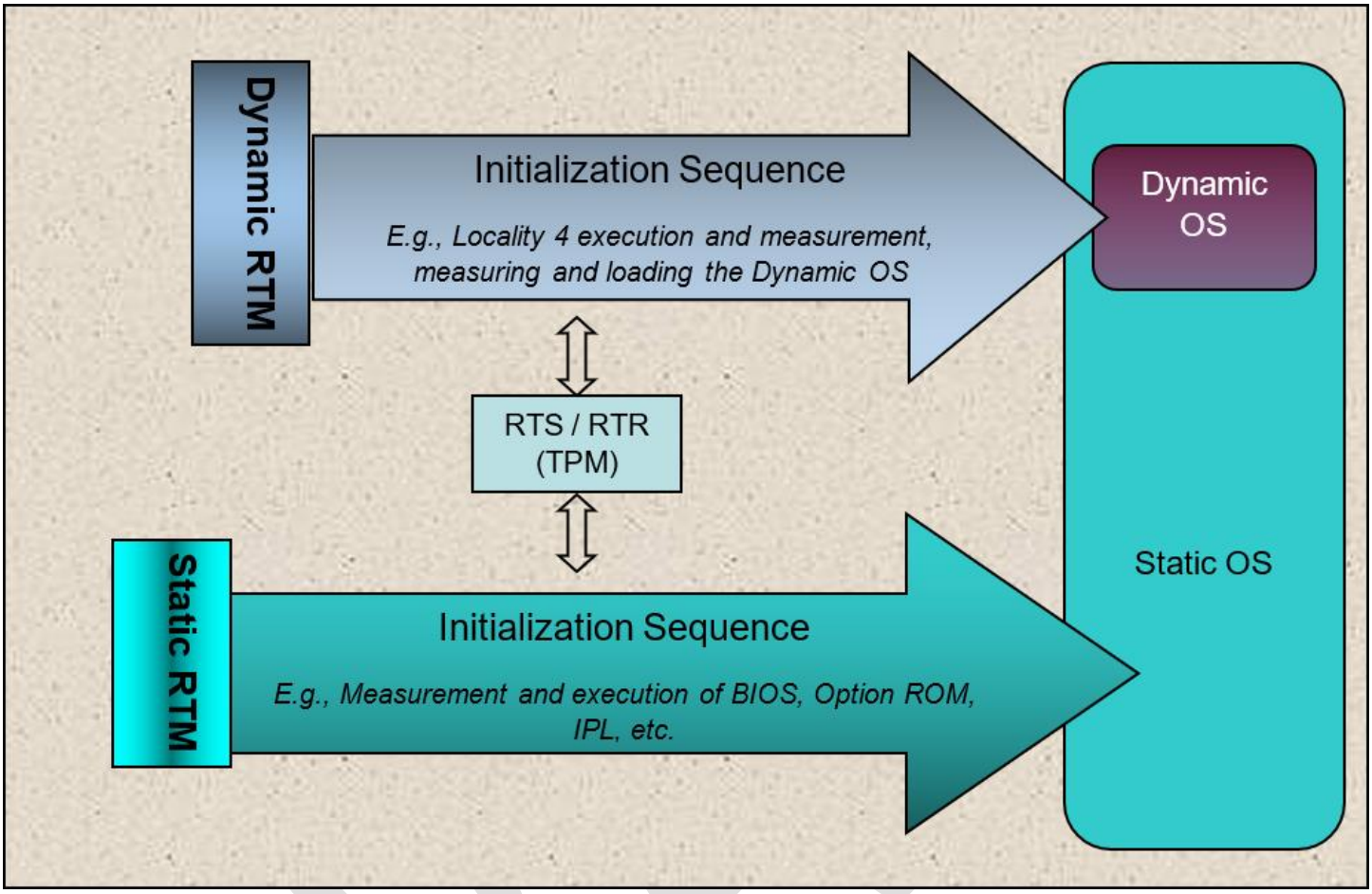


**Figure 2 Example of SRTM and DRTM Initialization Sequence**

#### 1.2.13.1 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 DRTM, SRTM or some subset of both.

An example: The Host Platform is used 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 SRTM'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 SRTM chain of trust) before proceeding to verify the security assertions of the DRTM.

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 DRTM and the SRTM is the RTR/RTS (TPM).
2. There is no direct relationship between or dependence on the chain of trust established by the DRTM and the chain of trust established by the SRTM. 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.

### 1.2.14 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:

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 – Platform Firmware 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 PC Client Platform TPM Profile for TPM 2.0, Section 3.8 (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 TPM2_Shutdown(STATE) 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**

### 1.2.15 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 clock. There is no requirement for the TPM to maintain this clock 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 the Measurement Process

**Start of informative comment**

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.

As a TPM in a PC Client system is usually a discrete chip that runs at lower clock speeds than the CPU and is connected to the platform via a slow bus, interaction with the TPM can be a platform bottleneck during the boot

process. Thus, this specification recommends minimizing the amount of data sent to the TPM for measurement and, once the initial measurement is made, making use of the CPU to perform hash computations of large amount of data.

**End of informative comment**

### 1.3.1 Usage and Optimization of Hash Functions

1. The Platform Firmware MAY use the TPM's Hash interface command, but SHOULD do so for as little data and as few times as possible.
2. The Platform Firmware SHOULD use its own, Host Platform CPU-based hashing function as early as possible and SHOULD continue to use it.

## 1.4 Terminology

**Start of informative comment**

The following terms are used as defined below throughout the document. All other terms are defined in the PC Client Implementation Specification.

**TPM Device Reset:** the assertion of the __TPM_INIT hardware signal.

**Platform Software:** the source of the command, which may be an operating system driver or an application.

**Platform Hardware:** platform components including chipsets and associated microcode, and microprocessors and associated microcode.

**SRTM:** code supplied by the platform manufacturer, as a subset of Platform Firmware that initializes and configures platform components and is the portion of Platform Firmware that defines the initial trust boundary.

**Operating System**, or **OS:** generic term for the system software and its collection of drivers and services that manages computer hardware and software resources.

**Static OS:** the operating system that is loaded during the initial boot sequence of the platform from its platform reset.

**Dynamic OS:** an operating system that is loaded by 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.

**Read:** a transaction where the calling entity requests and receives data from a specified register or buffer in the TPM.

**Write:** a transaction where the calling entity sends data to a register or buffer in the TPM.

**PC Client Platform Implementation Specification:** the combination of the *PC Client Platform Firmware Profile Specification*, the *TCG EFI Protocol Specification,* the *PC Client ACPI Specification*, and the PC Client Physical Presence Interface Specification.

**NUL:** indicates the ASCII null character, "\0". Used in this specification to indicate a null character.

**End of informative comment**

 

## 1.5 TCG Specification Dependency and Naming

### 1.5.1 Division of Documentation

**Start of informative comment**

The PC Client Specifications are divided into two documents:

1. The *PC Client Specific Platform TPM Profile for TPM 2.0* (PTP) discusses the specifics regarding the requirements of the TPM for PC Client but only the requirements for the TPM itself, not the requirements for a platform integrating the TPM. The PTP discusses the details of what interfaces and protocols are used to communicate with the TPM and the platform-specific set of requirements. The PTP includes the definitions of the items identified in the TPM Library specification as "Platform Specific" such as the minimum number of PCRs required and NV Storage available. The target audience for the PTP is the TPM manufacturers but platform manufacturers should review it as well.
2. This specification, the *PC Client Platform Firmware Profile,* specifies the requirements for the TPM as it is implemented on the platform. Issues such as TPM, platform and firmware provisioning, usage of TPM to record measurements of platform code, PCR mapping, functional interfaces, and interfaces are discussed. The target audience for this document is platform manufacturers.

The following TCG Specifications are referenced in this specification.

**End of informative comment**

### 1.5.2 "This" Specification

References to "this specification," unless contextually referencing a different antecedent, refer to the informative and normative comments contained in this document.

This specification defines only functional aspects of the concepts and implementation of a TPM, SRTM, 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.

### 1.5.3 Platform TPM Profile (PTP)

**Start of informative comment**

The PC Client Specific Platform TPM Profile for TPM 2.0 (PTP), Version 2.0, Revision 1.00 is the "companion" specification to this specification. The PTP 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**

### 1.5.4 TPM Library Specification

**Start of informative comment**

Refers to the TCG TPM Library Specification, Family 2.0 as released. The specification has four parts. Unless a specific part is referenced, this reference refers to all parts of the specification.

**End of informative comment**

### 1.5.5 TCG ACPI Specification

**Start of informative comment**

Refers to the TCG ACPI Specification for Family 1.2 and Family 2.0, Level 00 Revision .37, as released. The specification is managed by the server work group but is leveraged by PC Client platforms.

**End of informative comment**

 

### 1.5.6 TCG Physical Presence Specification

**Start of informative comment**

Refers to the TCG PC Client Physical Presence Interface Specification, Version 1.3, Revision 52 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 1.0

**End of informative comment**

### 1.5.8 TCG EFI Protocol Specification

**Start of informative comment**

Refers to the TCG EFI Protocol Specification for TPM Family 2.0 version 1.0 revision 0.9 or later.

**End of informative comment**

### 1.5.9 TCG Reference Integrity Manifest Information Model

**Start of informative comment**

Refers to the TCG Reference Integrity Manifest Information Model version 1.0 or later.

**End of informative comment**

## 1.6 External Specifications

This section lists references to external non-TCG specifications.

Advanced Configuration and Power Interface (ACPI), Version 2.0, http://www.acpi.info

Unified Extensible Firmware Interface Specification (UEFI), Version 2.4 (Errata B) or later, http://www.uefi.org

Microsoft Portable Executable and Common Object File Format Specification, Revision 6.0 or later, available at www.microsoft.com/whdc/ system/platform/firmware/PECOFF.mspx

Microsoft Windows Authenticode Portable Executable Signature Format, Version 1.0, Microsoft Corporation, March 21, 2008, http://download.microsoft.com/download/9/c/5/ 9c5b2167-8017-4bae-9fde-d599bac8184a/Authenticode_PE.docx

System Abstraction Layer (SAL), http://www.intel.com/design/itanium/downloads/245359.htm

U.S. National Institute of Standards and Technology (NIST) Special Publication 800-147 BIOS Protection Guidelines available at: http://csrc.nist.gov/publications/nistpubs/800-147/NIST-SP800-147-April2011.pdf

U.S. National Institute of Standards and Technology (NIST) Special Publication 800-155 BIOS Integrity Measurement Guidelines available at: http://csrc.nist.gov/publications/drafts/800-155/draft-SP800-155_Dec2011.pdf

DMTF Security Protocol and Data Model (SPDM) Specification 0.99a available at: https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_0.99a.pdf

## 1.7 Specification Conventions

**Start of informative comment**

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 format 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.
3. ASCII strings, unless otherwise indicated, are not NUL terminated.
4. Hexadecimal numbers are designated by '0x' preceding the number or 'h' following the number. Decimal numbers do not have the preceding 0x.
5. In some memory layout descriptions, certain fields are marked reserved. Firmware MUST initialize such fields to zero and ignore them when read. On an update operation, firmware MUST preserve any reserved field.

# 2 Host Platform Roots of Trust Requirements

## 2.1 Locality Support Requirements

### 2.1.1 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 may also have 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 PC Client Platform TPM Profile for TPM 2.0 for details about how these transitions are performed.

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

**Note:** Per section 15.1.4, Platform Firmware uses Locality 0 to access the TPM.

**End of informative comment**

1. In Pre-OS, Platform Firmware MUST transfer control to UEFI Drivers and Applications with the TPM in Locality 0.
2. UEFI Drivers and Applications MUST transfer control back to Platform Firmware with the TPM in Locality 0.

### 2.1.2 OS Environment

**Start of informative comment**

Because a Static Root of Trust for Measurement 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.

**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.

## 2.2 SRTM

### 2.2.1 Introduction of Concepts

**Start of informative comment**

The SRTM 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 SRTM 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 SRTM is the default RTM for all components because the SRTM measurements occur during boot by default. The

SRTM uses Locality 0 and the memory addresses associated with Locality 0. See the PC Client Platform TPM Profile for TPM 2.0 for definitions of locality and memory addresses.

**End of informative comment**

### 2.2.2 Initial TBB Control and Host Platform Reset

Host Platform Reset MUST cause the Host Platform to begin execution within the SRTM.

### 2.2.3 Static Root of Trust for Measurement (SRTM)

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

**Note:** The trust in the SRTM environment is based on the SRTM. The trust in all SRTM measurements is based on the integrity of the SRTM component.

Currently, in a PC, there are many types of SRTM architectures. An example is below.

#### 2.2.3.1 Firmware Boot Block SRTM

**Start of informative comment**

In this architecture, the Platform Firmware is composed of a Boot Block (SEC/PEI/IBB) and a UEFI firmware. Each of these is an independent component and each can be updated independent of the other. In this architecture, the Boot Block is the SRTM while the UEFI Firmware is not, but is a measured component of chain of trust.

**End of informative comment**

### 2.2.4 Transfer of Control from SRTM

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

## 2.3 Integrity Collection and Reporting

**Start of informative comment**

The Static PCRs, those that cannot be reset without a TPM_Init, are divided into two primary sets. The first set is designated for the Host Platform's pre-OS environment (PCR[0-7]) and the other designated for the Host Platform's Static OS (PCR[8-15]), see the PC Client Platform TPM Profile for TPM 2.0 for further information. PCR[6] 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 SRTM through the OS's Boot Manager Code. The definition of the Static OS PCRs is outside the scope of this specification and is the purview of the specific OS provider.

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 "Firmware 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 disable the TPM along with preventing the Platform Firmware 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.

This specification defines a list of measurements to be placed in each PCR. 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 properly, it is important that the sequence of the measurements be consistent between each platform reset when there are no security-related changes. Platform Firmware vendors 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.

With the introduction of UEFI systems, TCG produced the TCG PC Client EFI Protocol Specification. UEFI provides services which can be utilized by both the Platform Firmware and the Operating system. Two of these services which are critical to the correct operation of the integrity measurement and reporting system are the Get Event Log call (EFI_TCG2_GET_EVENT_LOG) and the Exit Boot Services call (EVT_SIGNAL_EXIT_BOOT_SERVICES). The Get Event Log call allows the OS to ask Platform Firmware to pass the platforms' TCG event log to the OS. The Exit Boot Services call (EBS) signals the end of Platform Firmware control of the boot process and the beginning of the OS control. Unfortunately, these two calls can be asynchronous, creating a situation where measurements can be made to the TPM and logged in the TCG Event Log after the OS has already received the Event Log. To address this, this specification adds a second instance of the event log, the Final Events Table, which is contained in an instance of a UEFI_CONFIGURATION_TABLE. This table will contain duplicate information for only those events which occur after the Get Event Log call.

**End of informative comment**

### 2.3.1 Collection and Reporting of Measurements

1. Platform Firmware 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.
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 SRTM MUST disable the TPM as specified in Section 6.1 TPM Visibility to the OS.
4. Unless TPM is hidden, integrity measurements made by the Platform Firmware that are extended MUST be extended into all allocated banks for a PCR, see Section 9.1 Introduction.
5. Integrity measurements by Platform Firmware that are extended into PCR[0-7] MUST be done only while the Host Platform is in the Pre-OS environment after starting from an Off state.
6. 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.
7. When the TPM is visible, Platform Firmware MUST log all measurements made to the TPM in the TCG Event Log. See Section 9.1 Introduction:
    a. Prior to Exit Boot Services and the call by the Operating System to get the Event TCG Event Log.
    b. After the first call to get the Event Log from Platform Firmware, all measurements made by Platform firmware MUST be reported in the Event Log and in the EFI_TCG2_FINAL_EVENTS_TABLE, as defined in the TCG EFI Protocol Specification v1.0 revision 9 Section 7 (Log Entries after Get Event Log Service).
8. When the TPM is hidden, Platform Firmware MUST cap PCR[0-7] as defined in Section 2.3.4 PCR Usage, but SHALL NOT log measurements in the TCG Event Log. See Section 9.1 Introduction0.
9. Integrity measurements made by the platform manufacturer for PCR[6] MAY occur at any time (Pre-OS or Post-Boot). See Section 2.3.4.7 PCR[6] – Host Platform Manufacturer Specific Measurements.
10. When the TPM is visible, but independent of TPM state, Platform Firmware MUST measure EV_SEPARATOR as defined in Section 9.4.1 Event Types. If the TPM is enabled, this event MUST be

 

entered into both copies of the event log. If the TPM is visible to the Operating System but the hierarchies are disabled, Platform Firmware MAY make other measurements as defined in this specification.

**Note:** EV_SEPARATOR delineates the point in the platform boot where Platform Firmware relinquishes control of the TPM measurement process. Measuring and extending EV_SEPARATOR enables an observer to know where Platform Firmware control ended. As an example, say that Platform Firmware controls a secret that is protected by the TPM. This secret must be protected from OS-resident agents. In this case, Platform Firmware would seal this secret to one or more TPM PCR such that only Platform Firmware could unseal it. A method to do this would be to calculate the value of the desired PCR prior to the EV_SEPARATOR event and seal to that value. Any attempt to unseal the PCR after this point would fail, as EV_SEPARATOR would alter the PCR value.

**End of informative comment**

11. While not all events described in Section 9.4.3 EV_ACTION Event Types are produced by every platform on every boot, when one is produced, platform Firmware MUST create the event indicated in Section 9.4.3 EV_ACTION Event Types.

### 2.3.2 Error Conditions

**Start of informative comment**

Some of these conditions include failure of the TPM or its code on its power on initialization test, presentation by Platform Firmware of an incorrect Startup Type, or the TPM2_Startup command originating from an incorrect locality. Unlike with TPM 1.2, there are multiple ways to handle these errors.

If the SRTM is unable to successfully initialize the TPM using the TPM2_Startup command, it is possible that later code could successfully issue the command and record false integrity measurements. To prevent this, the SRTM 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_RC_FAILURE.

This section applies for resume from S3 or normal boot.

Figure 3 shows the remediation steps the SRTM takes when initializing the TPM fails.

 

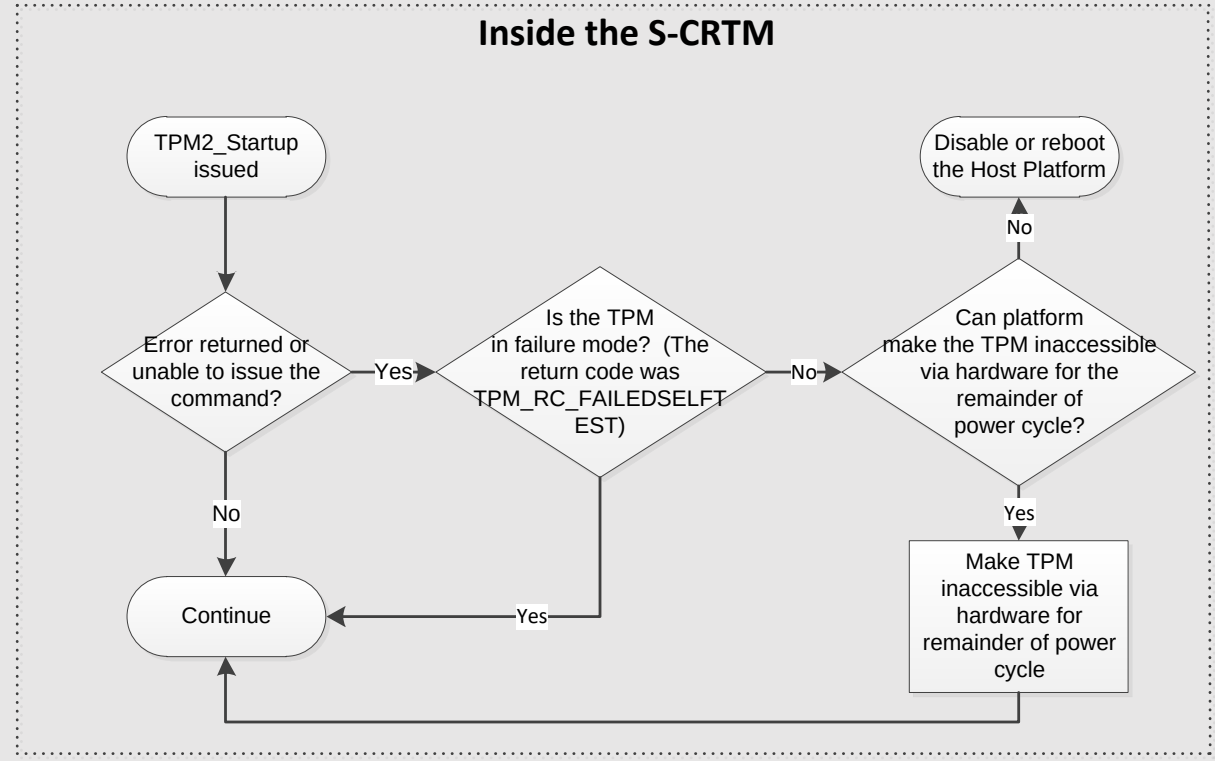


**Figure 3 SRTM remediation steps when initializing the TPM**

**End of informative comment**

#### 2.3.2.1 Errors during TPM Initialization

1. If the TPM responds to a TPM2_Startup command with TPM_RC_INITIALIZE, Platform Firmware MAY ignore the return code and continue to boot.
2. If the TPM responds to a TPM2_Startup command with TPM_RC_FAILURE, Platform Firmware SHALL:
   a. Reboot the platform and perform a TPM2_Startup (CLEAR), or
   b. With the exception of sending the TPM2_HierarchyControl command, perform the actions defined in Section 6.1 TPM Visibility to the OS for hiding the TPM.
3. If the TPM responds to a TPM2_Startup command with TPM_RC_VALUE or TPM_RC_LOCALITY, the Platform Firmware SHALL:
   a. Reboot the platform, perform a TPM2_Startup (CLEAR) and make all required measurements.
4. If the TPM responds to a TPM2_Startup command with any of the following error codes: TPM_RC_COMMAND_CODE, TPM_RC_UPGRADE or TPM_RC_REBOOT, the TPM has attempted to perform a firmware field upgrade and encountered an error, e.g. power removed during the upgrade. Platform Firmware SHALL:
   a. Restart the field upgrade process, OR
   b. Notify the user and, with the exception of sending the TPM2_HierarchyControl command, perform the actions in 6.1 TPM Visibility to the OS for hiding the TPM.
5. If the TPM responds to a TPM2_Startup with TPM_RC_BAD_TAG, Platform Firmware SHALL:
   a. Send a TPM_Startup (TPM 1.2) command.
      i. If the TPM returns with TPM_RC_SUCCESS, then a TPM 1.2 is present and the behavior is not defined in this specification.
      ii. If the TPM returns with TPM_RC_BAD_TAG, then the TPM is in Field Upgrade Mode. Platform Firmware SHALL perform the actions in normative 4.

#### 2.3.2.2 Errors Recording Measurements

1. If the measurement of the SRTM, POST BIOS or Embedded UEFI Drivers cannot be made due to a failure of the Platform Firmware, the PCR's MUST be capped by extending the digest of the value 00000001h to each PCR[0-7], and an EV_SEPARATOR event per Section 9.4.1 Event Types SHOULD be recorded in the event log.
2. The SRTM SHOULD log the error event to the event log.
3. 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 and transition to a "fail-safe" mode by performing one of these actions:
    a. Make the TPM interface inaccessible via hardware for the remainder of the power cycle;
    b. Reboot the Host Platform;
    c. Disable the Host Platform; OR
    d. Perform a vendor-specific action that is equivalent to one of the options above.

### 2.3.3 Boot Event and PCR Usage Model

**Start of informative comment**

The purpose of this section is to provide the model for PCR usage and for adding events to the Event Log. This entire section is an Informative comment, including Figure 4, Figure 5, and Figure 6.

- Figure 4 is an architecture drawing that shows the principle firmware and software components defined in the UEFI Specification – UEFI Boot Services, UEFI Runtime Services, and the UEFI OS Loader – and their relationship to the platform hardware and the OS software.
- Figure 5 shows the sequence of behaviors for the UEFI components during the platform boot process and the OS boot process.
- Figure 6 maps, in general, the behavioral components on a UEFI platform to the PCRs into which each type of behavioral component is measured.

Together, these three diagrams form the boot event and PCR usage model upon which the requirements in subsequent sections of this specification are based.

Figure 4, immediately below, is part of this Informative comment. This diagram illustrates the relationships of various components of a UEFI-specification compliant system that accomplish platform boot and OS boot.

 

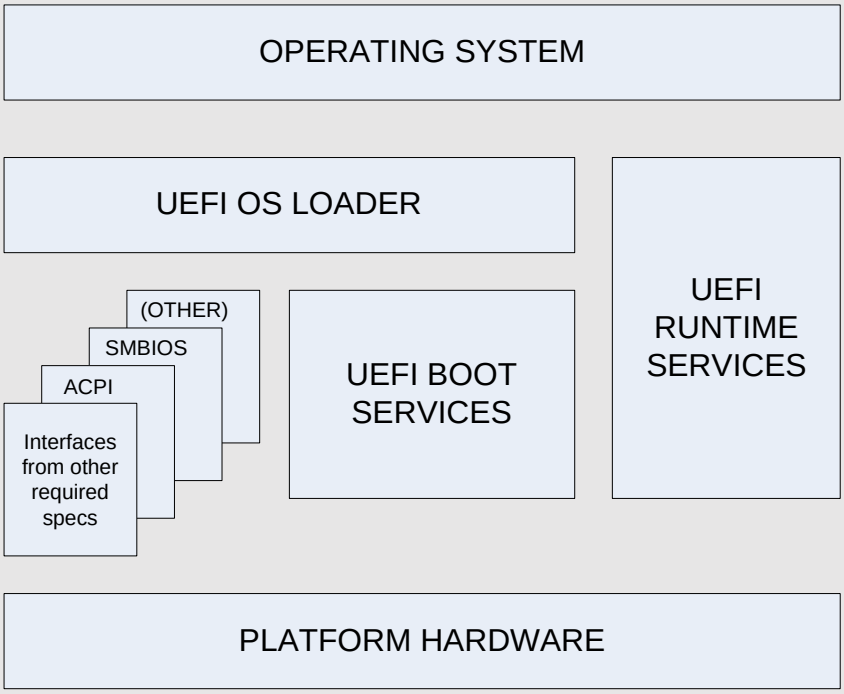


**Figure 4 UEFI Architecture**

Figure 5, immediately below, is part of this Informative comment and shows the sequence of behaviors for the UEFI components during the UEFI platform boot process and the UEFI OS boot process and, in general, the transitions where events are measured (labeled e0, e1, e2, and so on in the diagram).

In Figure 5, each one of the arrows with an “e” annotation represents a transition where a TCG Event is created; an un-annotated arrow does not represent a TCG event measurement. For example, the first measurement action, labeled “e0” is made by the platform initialization code.

Event “e0” describes the behavior of Platform Firmware from system board ROM that measures code contained in the rest of the platform UEFI firmware and that resides in the CRTM. This measuring agent may or may not contain the UEFI Boot and Runtime Services. In the case of a CRTM that does not contain UEFI Boot and Runtime Services, this measuring agent must then measure into PCR[0] the code that does contain UEFI Boot and Runtime Services. This initial code, if in CRTM, must measure its own version ID into PCR[0].

Event “e5” describes invocation of a UEFI application in the boot order list. This is typically an operating system loader. The event “e5” will have associated with it an event that measures the PE/COFF image into PCR[4] and the contents of the boot variable, namely the UEFI device path, into PCR[5]. For more information about this measurement action, see Section 7.2.2 Extending PCR[5].

This informative comment has given a couple of examples of measurement actions associated with particular platform boot transition events, but makes no attempt to describe the measurement events associated with all the transitions shown in Figure 5. To see all the measurement events associated with all the transitions shown in Figure 5, see Section 2.3.4 PCR Usage of this specification.

 

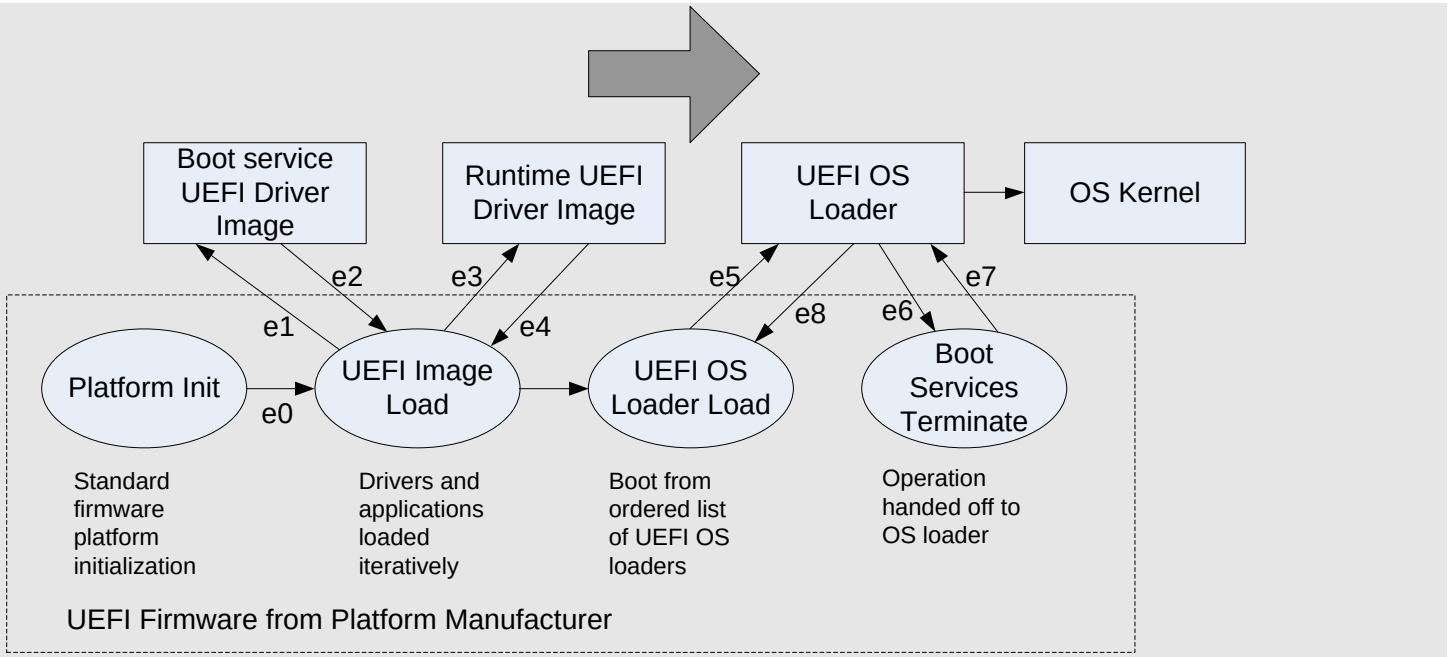


**Figure 5 UEFI Platform Boot Process**

Figure 6 PCR Mapping of UEFI Components, immediately below, shows a diagram illustrating the general scheme for PCR usage on a UEFI platform.

 

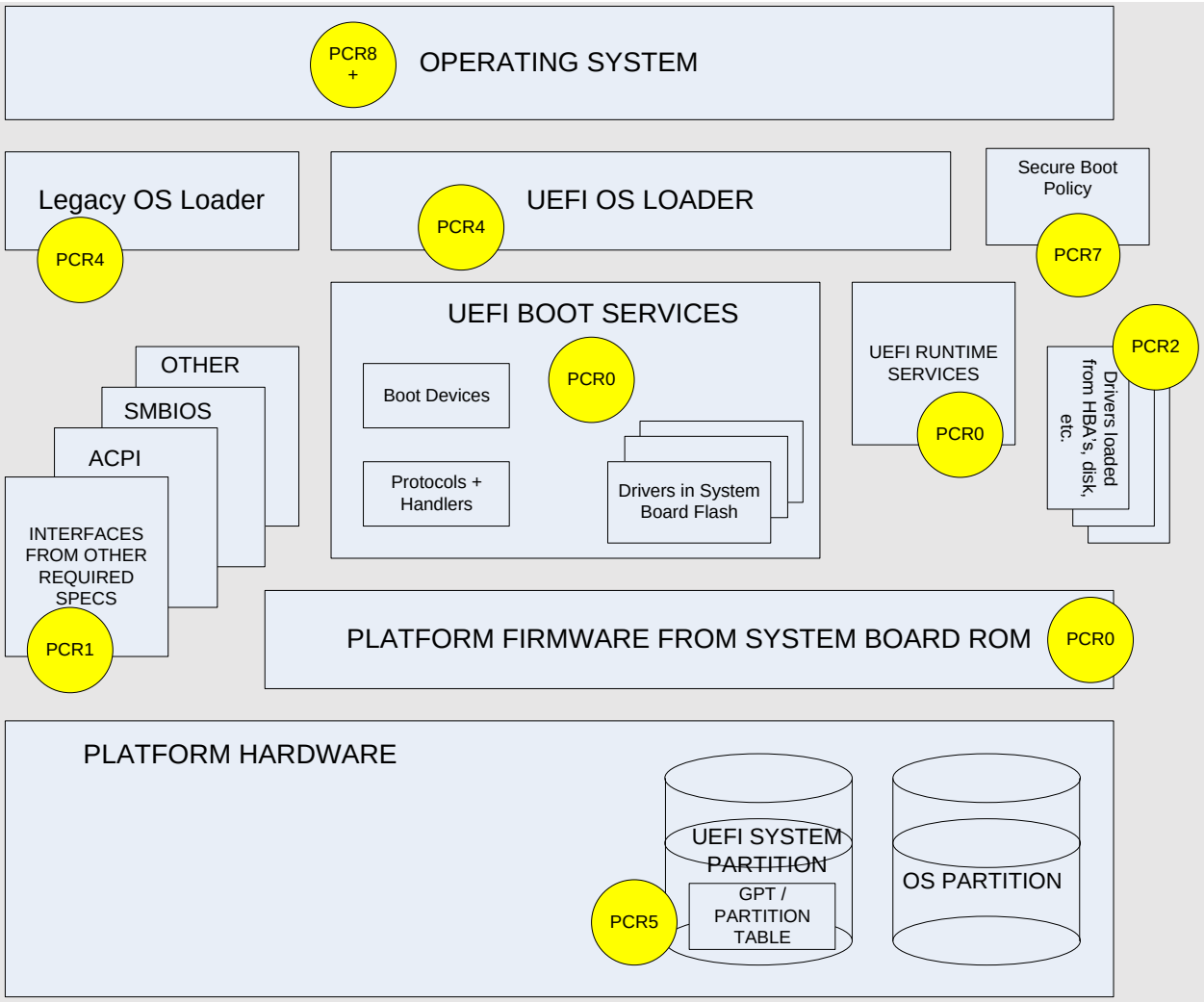


**Figure 6 PCR Mapping of UEFI Components**

**End of informative comment**

### 2.3.3.1 Measuring PE/COFF Image Files

**Start of informative comment**

These images are measured in their memory-mapped store (ROM for execute-in-place and system memory for loaded images).

The measurement must be made prior to the application of any fix-ups or relocations.

An example of a PE/COFF image file is a UEFI OS loader, which would be an OS subsystem type of IMAGE_SUBSYSTEM_EFI_APPLICATION, IMAGE_SUBSYSTEM_EFI_BOOT_SERVICE_DRIVER, and IMAGE_SUBSYSTEM_EFI_RUNTIME_DRIVER (as per PE/COFF specification section 3.4.2).

For UEFI images, the data structure used in the TCG_PCR_EVENT2.event field will include but is not limited to "where" the image came from, namely its UEFI device path, and the address in memory for the shadowed PE image (see the definition of the UEFI_IMAGE_LOAD_EVENT structure, in the mandatory statements below).

The image in physical memory will ultimately have relocations applied (i.e., fix-ups). The event log will detail the load location of the image, so it will be possible to undo relocations by referring to the on-media image.

The Microsoft Authenticode Portable Executable Signature Format Specification identifies which PE/COFF image section types UEFI measurement agents must measure and which types of PE/COFF image sections are ignorable. What "measure before applying relocations" described above means in practice is that the UEFI implementation will perform "LoadImage()" actions (e.g., copying PE/COFF to memory, etc.), measurement, and then relocation application, and finally, the UEFI service "StartImage()." As such, UEFI implementations of these services must punctuate their flow with this measurement action.

**End of informative comment**

1. When measuring a PE/COFF image, the TCG_PCR_EVENT2 structure Event field MUST contain the UEFI_IMAGE_LOAD_EVENT structure defined in Section 9.2.3 UEFI_IMAGE_LOAD_EVENT Structure.

### 2.3.4 PCR Usage

**Start of informative comment**

PCR[0-15] represent the Host Platform's static RTM and are associated with Locality 0. 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 OS-resident). Throughout the platform 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**

**Table 1 PCR Usage**

| PCR Index | PCR Usage |
|---|---|
| 0 | SRTM, BIOS, Host Platform Extensions, Embedded Option ROMs and PI Drivers |
| 1 | Host Platform Configuration |
| 2 | UEFI driver and application Code |
| 3 | UEFI driver and application Configuration and Data |
| 4 | UEFI Boot Manager Code (usually the MBR) and Boot Attempts |
| 5 | Boot Manager Code Configuration and Data (for use by the Boot Manager Code) and GPT/Partition Table |
| 6 | Host Platform Manufacturer Specific |
| 7 | Secure Boot Policy |
| 8-15 | Defined for use by the Static OS |
| 16 | Debug |

 

| PCR Index | PCR Usage |
|---|---|
| 23 | Application Support |

1. Firmware on a PC Client Platform compliant with this specification SHALL measure all normative items using a tagged Hash structure for each PCR bank enabled in the platform's TPM. See Section 9.2 TCG Defined Structures.

#### 2.3.4.1 PCR[0] – SRTM, POST BIOS, and Embedded Drivers

**Start of informative comment**

The SRTM may measure itself or may be measured by an H-CRTM event extended to PCR[0]. The SRTM must measure to PCR[0] any portion of the PEI Code, including Manufacturer Controlled Embedded Drivers, 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 SRTM PCR values can be trusted and therefore are outside the chain of trust. It is therefore necessary to invalidate all Host Platform SRTM PCRs as described in Section 2.3.2 Error Conditions.

Because the measurement of the early Platform Firmware is typically done within a resource-constrained environment (for example, the SRTM, likely the PEI Code) the event log corresponding to the Extend event may not be created at the time the TPM2_PCR_Extend is performed. It is acceptable to have the Platform Firmware generate the corresponding event log entry, reconstructing it from information (for example, the value that was extended) passed to it from PEI or earlier code. If this procedure is used, the Platform Firmware must be sure the entries within the event log are properly sequenced (for example, represent the same order as the sequence of TPM2_PCR_Extend operations).

**Measurement of Non-Host Platforms**

If a Non-Host Platform cannot be reliably detected by the Platform Firmware and measured into PCR[0], the existence of that Non-Host Platform should be indicated in the Host Platform Certificate. If a Non-Host Platform can be modified by an entity other than the Platform Firmware, it is measured in PCR[2]

**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 UEFI Driver". 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 UEFI Drivers and PCR[0], PCR[2] and PCR[3]**

Manufacturer Controlled UEFI Drivers may be Drivers for devices that the platform manufacturer physically soldered to the motherboard or Drivers whose code is embedded within a firmware image. Embedded Drivers 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 Drivers, or UEFI Drivers, or Applications associated with devices in adapter slots may be added, removed, or updated by a platform owner and their code is measured in PCR[2].

Some UEFI Applications may use paging or other techniques to load and execute code that was hidden from the Platform Firmware when measuring the visible portion of the Application. It is the responsibility of the UEFI Application to measure this code prior to executing any portion of that hidden Option ROM code in PCR[2].

Hidden UEFI application code measurements are always placed in PCR[2]. UEFI Application configuration measurements are always placed in PCR[3]. Recording Manufacturer and Non-Manufacturer controlled hidden UEFI Driver and Application code consistently in PCR[2] and PCR[3], respectively, removes the need for a UEFI Application to know how it was packaged in a system. (For example, as a Manufacturer Controlled Embedded Driver versus as an Application 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 measurement of code of 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 Driver 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 OS-Present state of the platform. This PCR, along with PCR[5], identifies the initial OS loader.

In general, measure the following types of code into PCR[0] (as shown in Figure 6 PCR Mapping of UEFI Components):

- Platform Firmware from system board ROM that either contains or measures the UEFI Boot Services and UEFI Run Time Services that are loaded from firmware
- Embedded UEFI Drivers wholly contained within the Platform Firmware on the system board ROM
- EFI Boot Services in the UEFI System Table created in memory by Platform Firmware
- EFI Runtime Services in the UEFI System Table

A UEFI platform measures firmware integrated in the system board into PCR[0]. These are code modules known to be distributed by the Host Platform manufacturer. For a UEFI platform, this includes but is not limited to measuring:

- Version of the base firmware that either contains or measures the UEFI Boot Services (shown as the Platform Init code module of the UEFI Boot Manager in Figure 4 UEFI Architecture)

**End of informative comment**

**Entities that MUST be logged in the Event Log if the TPM is enabled:**

1. The event type EV_NO_ACTION Specification ID Event. See Section 9.4.5.1 Specification ID Version Event. Note: This event is not extended.
2. The VendorID and ReferenceManifestGUID of the Platform Firmware which either contains or measures the UEFI Boot Services and UEFI Run Time Services using EV_NO_ACTION event ReferenceManifestEvent. This event is only required if the platform supports NIST SP800-155. This event is not extended. See Section 9.4.5.2 Firmware Integrity Measurement Reference Manifest Event.

**Entities that MUST be measured if the TPM is enabled:**

1. The event type EV_NO_ACTION Startup Locality Event to record the locality from which the TPM2_Startup command was sent, if the Locality sending the TPM2_Startup command is Locality 3. See Section 9.4.5.3 Startup Locality Event.
2. If an H-CRTM event occurred, the H-CRTM event SHALL be measured using the event type EV_EFI_HCRTM_EVENT. The event data SHALL be the string “HCRTM”. See Section 9.4.1 Event Types.
3. The SRTM’s version identifier, using the event type EV_S_CRTM_VERSION. See Section 9.4.1 Event Types.
4. 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 9.4.1 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 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 be the event field string of “POST CODE” in all caps).
        ii. Embedded SMM code and the code that sets it up (SHOULD be the event field string of “SMM CODE” in all caps).
    b. BIS code (excluding the BIS Certificate) (SHOULD be the 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 be the event field string of “ACPI DATA” in all caps.)
    d. Manufacturer Controlled Embedded UEFI modules/drivers as a binary image whose release and update is controlled by the Host Platform Manufacturer (SHOULD be the event field string of “Embedded UEFI Driver”).
5. EFI drivers embedded in system ROM by the platform manufacturer using event type EV_EFI_RUNTIME_SERVICES_DRIVER.
6. The Platform Firmware that either contains or measures the UEFI Boot services and UEFI Run Time Services using event type EV_EFI_PLATFORM_FIRMWARE_BLOB2, see Section 9.2.5 UEFI_PLATFORM_FIRMWARE_BLOB Structure Definition.
7. Platform Firmware MUST attempt to detect and measure the presence of any Non-Host Platform. If Platform Firmware 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 9.4.1 Event Types.
8. Per Section 2.3.2 Error Conditions, if an error occurs, the digest of the value 00000001h MUST be extended in PCR[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 9.4.1 Event Types.

9. Per Section 7.2.4 Measuring OS Boot Events, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call. See Section 9.4.1 Event Types.
10. In some Firmware update scenarios during resume from S2 or S3, per requirements in Sections 7.3.7 S2 (Sleep) to S0 (Working) and 7.3.9 S3 (Sleep) to S0 (Working), a digest of the value 00000001h MAY be extended in PCR[0] to invalidate PCR[0].

**Entities that MAY be measured if the TPM is enabled:**

1. The SRTM itself using event type EV_S_CRTM_CONTENTS. See Section 9.4.1 Event Types.

**Entities that SHOULD be measured if the TPM is enabled:**

1. Components within Non-Host Platforms (e.g., firmware not intended to be executed by Host Platform's CPU) that can only be updated by Platform Firmware and 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 9.4.1 Event Types.

**Entities that MUST be measured if the TPM is disabled:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and the matching EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call.

**TPM device is hidden:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] prior to the first invocation of the first Ready to Boot call. Platform Firmware SHALL NOT place an event in the event log.

**Method for measurement:**

The SRTM performs these measurements as follows:

1. Measure the SRTM's version identifier.
2. Measure the code to which the SCRTM is transferring control.
3. Platform Firmware 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, e.g. the Specification Version event, the Startup Locality event and the H-CRTM event.
4. The remaining measurements MAY be performed in any order, except for the EV_SEPARATOR prior to Ready to Boot 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 9.4.5.1 Specification ID Version Event. Note: This event is not extended.

### 2.3.4.2 PCR[1] – Host Platform Configuration

**Start of informative comment**

In general, measure into PCR[1] the data that is associated with the code that has been measured into PCR[0].

For performance reasons, some implementations may combine CPU microcode updates with a Firmware 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 Platform Firmware 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 memory slots exist on the platform and how many of these 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 setting.

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.

For example, for a UEFI server platform, this includes but is not limited to

- SAL system table
- SMBIOS Tables

PCR[1] also contains Boot Policy measurements. In order to record the alternate boot behaviors of a UEFI system, the UEFI Platform Firmware will measure the UEFI Boot#### variables and the BootOrder Variable into PCR[1]. These boot variables are just device paths. A description of the device paths and variables can be found in the UEFI Specification.

**End of informative comment**

**Entities that MUST be measured if the TPM is enabled:**

The following entities MUST always be measured. The platform manufacturer MUST NOT provide firmware configuration options that permit disabling the following required measurements:

1. If Platform Firmware loads a CPU microcode update, it MUST be measured, using event type EV_CPU_MICROCODE. See Section 9.4.1 Event Types. Exception: Alternatively, CPU microcode updates MAY be measured in PCR[0] as part of a EV_POST_CODE event.
2. EFI Boot#### and UEFI BootOrder variables MUST be measured using event type EV_EFI_VARIABLE_BOOT. See Section 9.2.6 Measuring UEFI Variables.
3. If the Host Platform supports selection 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 9.4.1 Event Types.
4. If the system supports UEFI 2.5 or later and DeployedMode is enabled, the following additional variables MUST be measured into PCR[1]:
   a. The DeployedMode variable value. The Event Type SHALL be EV_EFI_VARIABLE_DRIVER_CONFIG and the Event value shall be the value of the UEFI_VARIABLE data structure.
   b. The AuditMode variable value. The Event Type SHALL be EV_EFI_VARIABLE_DRIVER_CONFIG and the Event value shall be the value of the UEFI_VARIABLE data structure.
   c. If the platform supports changing either of these UEFI policy variables after they are initially measured in PCR[1],before ExitBootServices() has completed and the platform does not support UEFI 2.5 or later, the platform MUST be restarted. If the platform supports UEFI 2.5 or later and supports changing any of the UEFI Secure Boot policy variables after they are initially measured in PCR[1] and before ExitBootServices() has completed, the platform MAY be restarted OR the variables MUST be remeasured into PCR[1].
5. SMBIOS structures that contain static configuration information (e.g. Platform Manufacturer Enterprise Number assigned by IANA, platform model number, Vendor and Device IDs for each SMBIOS table) that is relevant to the security of the platform MUST be measured using the event type EV_EFI_HANDOFF_TABLES as described in Section 9.4.1 (Event Types).
6. Prior to entering a Platform Firmware Setup Utility, if the Host Platform does not unconditionally perform a Host Platform Reset upon exiting a Pre-OS Setup Utility, the platform MUST measure the EV_ACTION event “Entering ROM Based Setup”. See Section 9.4.3 EV_ACTION Event Types. (Note: This is only for Platform Firmware Setup Utilities as opposed to entering an Option ROM-Based Setup Utility that is recorded in PCR[3].)
7. Per Section 2.3.2 Error Conditions, if an error occurs, the digest of the value 00000001h MUST be extended in PCR[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 9.4.1 Event Types.
8. Per Section 7.2.4 Measuring OS Boot Events, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call. See Section 9.4.1 Event Types.

**Entities that SHOULD be measured if the TPM is enabled:**

1. EFI Setup Variables which contain security-relevant configuration data, including Setup Admin password settings and Setup User password settings, that is not measured elsewhere SHOULD be measured. If measured, the event MUST be recorded in the event log using EV_PLATFORM_CONFIG_FLAGS or EV_EFI_VARIABLE_DRIVER_CONFIG. See Section 9.4.1 Event Types.
2. Security-relevant CMOS data and security-relevant configuration data stored in Platform NVRAM and in effect when the platform boots SHOULD be measured. If measured, these events MUST use the event type EV_EFI_HANDOFF_TABLES as described in Section 9.4.1 Event Types. Sensitive data like passwords MUST be omitted from the NVRAM data because it could be placed unencrypted in the TCG event log.
3. Table of devices, SHOULD be measured using event type EV_TABLE_OF_DEVICES described in Section 9.4.1 Event Types.

 

This event allows Platform Firmware 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. Platform Firmware MAY create multiple entries of this event or MAY choose to encapsulate all the data into a single entry.

4. Non-Host Platform configuration information. If the system contains a Non-Host Platform that can only be updated by Platform Firmware, Platform Firmware SHOULD measure the configuration and MUST use the event type EV_NONHOST_CONFIG to record any security-relevant configuration information or data. See Section 9.4.1 Event Types.

**Entities that MAY be measured if the TPM is enabled:**

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

1. ESCD, using event type EV_EFI_HANDOFF_TABLES as described in Section 9.4.1 Event Types.
2. Other tables MAY be measured using event type EV_EFI_HANDOFF_TABLES.
3. EFI Variables that impact system configuration MAY be measured using event type EV_EFI_VARIABLE_DRIVER_CONFIG.
4. If Platform Firmware 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 Platform Firmware administrator clears the notification. See Section 9.4.3 EV_ACTION Event Types.

**Entities that MUST be measured if the TPM is disabled:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call.

**TPM device is hidden:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] prior to the first invocation of the first Ready to Boot call. Platform Firmware SHALL NOT place an event in the event log.

**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, as they would prevent sealing to PCR 1 with a common configuration in a fleet of devices.
3. The BIS Certificate if it contains privacy-sensitive information.

**Method for measurement:**

Platform Firmware 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 Ready to Boot invocation, which MUST be last. While these events may be measured in any order, the selected order MUST be maintained across boot cycles.

2. Where possible, these measurements SHOULD occur prior to measuring Option ROMs.

#### 2.3.4.3 PCR[2]-UEFI Drivers and UEFI Applications

**Start of informative comment**

EFI Drivers and Applications contained on Host Platform adapters are measured by Platform Firmware to PCR[2].

For a UEFI platform, this includes but is not limited to

- EFI Drivers loaded from adapter cards
- EFI Applications loaded from adapter cards (e.g., setup utility for RAID controller)
- EFI Applications loaded from external storage media (system or peripheral Flash for example) via embedded option ROM contained within the UEFI firmware on the system board. An example of this would be an embedded UEFI driver that loads more code from an external location, such as a hidden partition on a drive. An example would be an embedded driver that pages the external code in to memory, and thus the external code is not visible to LoadImage. If the embedded driver does not use the LoadImage service, it should perform all the actions that the LoadImage service would do.
- Drivers loaded from HBA's/disks and external storage media (as shown in Figure 6 PCR Mapping of UEFI Components).
- Non-Manufacturer Controlled Embedded UEFI Drivers and Applications 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 Application Code**

The PC Client Implementation provided a mechanism for adapters with Option ROMs to have hidden code that they were responsible for measuring. This specification does not support this behavior for UEFI adapters. All code should be measured by the LoadImage service.

**Measuring Non-Host Platforms**

If the platform contains a Non-Host Platform which can be updated by an entity other than the Platform Firmware, it is measured in PCR[2].

**Interpreting UEFI Driver Measurements**

The measuring agent is always the Platform Firmware (that is, code measured into PCR[0], even in the case where a measuring agent not measured into PCR[0] "requests" the program load, such as a UEFI Driver trying to load another UEFI Driver or Application). The UEFI Driver will use the UEFI LoadImage service. Since measurement occurs between shadowing of PE sections and the application of relocations, only the LoadImage service can call the TCG UEFI Protocol function TCG_HashLogExtendEvent at the appropriate time.

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

In general, the system measures into PCR[2] firmware code modules added to the Host Platform by a Value Added Retailer (VAR), the platform owner, or some unknown entity. The source of these firmware code modules measured into PCR[2] is not the Host Platform manufacturer.

**Note:** Components measured into PCR 2 and 3 may be updatable during OS runtime. As such, measurements of these components made during the platform boot process, e.g SPDM measurements, may not reflect the firmware running after ExitBootServices(). If this occurs the measurements in PCR 2 and 3 may

become stale until the next boot. If there is a policy that depends on PCR 2 and 3 values, the policy needs to account for this behavior.

**End of informative comment**

**Entities that MUST be measured if the TPM is enabled:**

1. EFI Boot Services drivers from add-in adapters or loaded by the driver in an add-in adapter MUST be measured using event type EV_EFI_BOOT_SERVICES_DRIVER. See Section 9.4.1 Event Types.
2. EFI Run Time drivers from add-in adapter or loaded by add-in adapter MUST be measured using event type EV_EFI_RUNTIME_SERVICES_DRIVER. See Section 9.4.1 Event Types.
3. Components within Non-Host Platforms (e.g., firmware not intended to be executed by the Host Platform's CPU) that can be updated by entities other than Platform Firmware and 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_NON_HOST_CODE. See Section 9.4.1 (Event Types).
4. Per Section 2.3.2 Error Conditions, if an error occurs, the digest of the value 00000001h MUST be extended in PCR[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 9.4.1 Event Types.
5. Per Section 7.2.4 Measuring OS Boot Events, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call. See Section 9.4.1 Event Types.

**Entities that SHOULD be measured if the TPM is enabled:**

1. Platform Firmware SHOULD measure the firmware of add-in devices that support the SPDM "GET_MEASUREMENTS" functionality using event type EV_EFI_SPDM_FIRMWARE_BLOB. See Sections 0(DEVICE_SECURITY_EVENT_DATA Structure) and Section 9.4.1 Event Types. **Note:** This measurement may be made mandatory in future revisions of this specification.

**Entities that MUST be measured if the TPM is disabled:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call.

**TPM device is hidden:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] prior to the first invocation of the first Ready to Boot call. Platform Firmware SHALL NOT place an event in the event log.

**Method for measurement:**

1. Platform Firmware MUST measure the event EV_EFI_RUNTIME_SERVICES_DRIVER or EV_EFI_BOOT_SERVICES_DRIVER for each UEFI Driver, non-bootable applications, Non-Manufacturer Controlled Embedded UEFI Driver, or Non-Manufacturer Controlled Embedded non-bootable applications prior to initializing the module.
2. Platform Firmware MUST measure the event type EV_POST_CODE for each Manufacturer Controlled Embedded UEFI Drivers (in PCR[0], see Section 2.3.4.1 PCR[0] – SRTM, POST BIOS, and Embedded Drivers prior to initializing the Manufacturer Controlled Embedded UEFI Driver.
3. The visible portion of UEFI Drivers and Applications measured in PCR[2] by Platform Firmware 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 adapters containing the Driver or Application is unchanged. Platform Firmware MAY measure all UEFI Drivers and Applications 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.
**Note:** Changing the adapter slot in which a UEFI Driver or Application is inserted may change the order of measurements.

4. Repeat until all UEFI Drivers and Applications are measured and executed.
5. When initialized, the UEFI Drivers and Applications MUST measure their "hidden" code prior to executing it.
6. The EV_SEPARATOR event prior to Ready to Boot invocation MUST be measured last.

#### 2.3.4.4 PCR[3] – EFI Driver/Application Configuration

**Start of informative comment**

This section contains the PCR[3] measurement requirements for a UEFI platform.

In general, entities measure into PCR[3] data that is associated with code modules that are measured into PCR[2]. For example, on a UEFI platform, this includes but is not limited to UEFI variables defined and managed by drivers. See section 9.2.4 Measuring Industry Standard Tables and Data Structures.

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

If a UEFI Application has a Pre-OS Setup Utility, it needs to record the event type EV_ACTION event "Entering ROM Based Setup" in PCR[3] prior to entering the utility.

If a UEFI Application Pre-OS Setup Utility permits the user to change 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 UEFI Application may be designed to permit the configuration changes to occur before the UEFI Application records the configuration data in PCR[3]. This may allow the Pre-OS Setup Utility to avoid needing to perform a Host Platform Reset if the Option ROM configuration is changed.

**Note:** Components measured into PCR 2 and 3 may be updatable during OS runtime. As such, measurements of these components made during the platform boot process, e.g SPDM measurements, may not reflect the firmware running after ExitBootServices(). If this occurs the measurements in PCR 2 and 3 may become stale until the next boot. If there is a policy that depends on PCR 2 and 3 values, the policy needs to account for this behavior.

**End of informative comment**

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

**Entities that MUST be measured if the TPM is enabled:**

1. If any configuration data exists in a UEFI Variable for a UEFI Driver or Application or the adapter that hosts the UEFI Driver, before that configuration data is used, it MUST measure the configuration data specific to the device using event EV_EFI_VARIABLE_DRIVER_CONFIG. See Section 9.4.1 Event Types.
2. Prior to entering an UEFI Application-Based Setup Utility, if the Host Platform does not unconditionally perform a Host Platform Reset upon exiting a Pre-OS ROM-Based Setup Utility, the platform MUST measure the EV_ACTION event "Entering ROM Based Setup". See Section 9.4.3 EV_ACTION Event Types. (Note: This is only for UEFI Application-Based Setup Utilities versus entering a ROM-Based Setup Utility which is recorded in PCR[1].)
3. Non-Host Platform configuration information. If the system contains a Non-Host Platform which can be updated by entities other than Platform Firmware, Platform Firmware SHOULD measure the configuration

and MUST use the event type EV_NONHOST_CONFIG to record any security-relevant configuration information or data. See Section 9.4.1 Event Types.

4. Per Section 2.3.2 Error Conditions, if an error occurs, the digest of the value 00000001h MUST be extended in PCR[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 9.4.1 Event Types.
5. Per Section 7.2.4 Measuring OS Boot Events, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call. See Section 9.4.1 Event Types.

**Entities that SHOULD be measured if the TPM is enabled:**

1. Platform Firmware SHOULD measure the firmware configuration, if present, of add-in devices that support the SPDM "GET_MEASUREMENTS" functionality using event type EV_EFI_SPDM_FIRMWARE_CONFIG. See Sections 9.2.7 DEVICE_SECURITY_EVENT_DATA Structure and 9.4.1 Event Types. **Note:** This measurement may be made mandatory in future revisions of this specification.

**Entities that MUST be measured if the TPM is disabled:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call.

**TPM device is hidden:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] prior to the first invocation of the first Ready to Boot call. Platform Firmware SHALL NOT place an event in the event log.

**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, as they would prevent sealing to PCR 1 with a common configuration in a fleet of devices.

**Method for measurement:**

The UEFI Application performs these measurements as follows:

1. Measures the event EV_EFI_VARIABLE_DRIVER_CONFIG, see Section 9.4.1 Event Types.
2. The remaining measurements MAY be performed in any order, except for the EV_SEPARATOR prior to Ready to Boot invocation, which MUST be last.

#### 2.3.4.5 PCR[4] – Boot Manager 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 Boot Manager Code that is loaded and executed from the device. If a boot from one device fails, measurements in PCR[4] record the attempt to boot the next device or boot path. If Boot Manager Code returns control back to Platform Firmware, each subsequent execution of Boot Manager Code is separately measured.

 

If boot fails for one device, it often returns execution control back to Platform Firmware so Platform Firmware may boot another device. Ultimately the boot process reflected in PCR[4] measurements may contain code from multiple boot devices. Platform Firmware 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 subsequent events, making it seem as if the malicious code failed to run.

If the boot device or the Boot Manager 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 Boot Manager Code that BIOS decides to load and measure into PCR[4] may contain code that is part of the OS environment. Boot Manager should record code to which it transfers control into PCR[4] or PCRs allocated for the OS. Boot Manager 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 Boot Manager 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 Manager 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 Platform Firmware to determine if 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 driver is measured in PCR[0]. The boot attempt may use a Platform Firmware 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 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 of the TCG PC Client Implementation Specification for Conventional BIOS, 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 Platform Firmware configuration or design, Platform Firmware records the event type EV_OMIT_BOOT_DEVICE_EVENTS in PCR[4]; otherwise, the event is not recorded.

**Boot Device Types**

This section of the UEFI Platform Specification contains PCR usage requirements for the transition from the UEFI Boot Manager to the OS Loader, shown in Figure 5 UEFI Platform Boot Process, above. Note that the term ‘EFI OS Loader Load’ is a label for a behavior, not necessarily a label for a code module.

This section contains the PCR[4] measurement requirements for a UEFI platform.

In general, entities measure into PCR[4] code modules that could transition the platform from the pre-OS state to the OS-present state. Measure UEFI applications into PCR[4]; for a UEFI platform, UEFI applications include but are not limited to:

- Pre-OS diagnostics
- EFI OS Loader

The measurement values in PCR[4] must be consistent and repeatable across UEFI boots.

In the table below, 'measuring entity' means code in the CRTM or code measured into PCR[0] which provides image loading capability and invokes the measurement agent (the 'platform code').

The 'measured object' is a PE/COFF image; see Section 9.2.3 UEFI_IMAGE_LOAD_EVENT Structure, for a definition of the methodology for measuring PE/COFF images on a TCG UEFI platform.

The measuring agent is always the Platform Firmware (that is, code measured into PCR[0]). The measuring agent will typically measure the measurement object code as part of the UEFI LoadImage Service. Since measurement occurs between shadowing of PE sections and the application of relocations, only the UEFI LoadImage Service can call the TCG UEFI Protocol function TCG_HashLogExtendEvent at the appropriate time.

This model purposely precludes pre-OS agents from loading and invoking code outside the UEFI LoadImage Service. Only the PCR[4] application, such as the UEFI OS Loader, can use a different code load and measurement methodology, but the target PCR for this action will be in the range PCR[8] and above and outside the scope of this specification.

**End of informative comment**

**Entities that MUST be measured if the TPM is enabled:**

1. If the Platform Firmware is configured or designed to not record each attempt to boot a device, an EV_OMIT_BOOT_DEVICE_EVENTS event MUST be measured once. See Section 9.4.1 Event Types.
2. Per Section 7.2.4 Measuring OS Boot Events, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call. See Section 9.4.1 Event Types.
3. When Platform Firmware transfers control to the Boot Manager, firmware MUST record the EV_ACTION event "Returned Ready to Boot". See Section 9.4.3 EV_ACTION Event Types.
4. For each boot attempt, if the EV_OMIT_BOOT_DEVICE_EVENTS event was not recorded, Platform Firmware MUST record the EV_EFI_ACTION event "Calling EFI Application from Boot Option" per Section 9.4.4 EV_EFI_ACTION Event Types when attempting to boot a device. See Section 7.2.4 Measuring OS Boot Events for specific details on how measurements are done for different boot devices and if applicable corresponding PCR[5] measurements.
5. For the UEFI application code PE/COFF image described by the boot variable, Platform Firmware MUST record the EV_EFI_BOOT_SERVICES_APPLICATION into PCR[4]. See Section 7.2.4 Measuring OS Boot Events.
6. Additional pre-OS environment code that is loaded by UEFI Applications using event EV_COMPACT_HASH. See Section 9.4.1 Event Types.
7. If a boot device returns control back to the Boot Manager, Platform Firmware MUST record the EV_EFI_ACTION event "Returning from EFI Application from Boot Option". See Section 9.4.3 EV_ACTION Event Types.
8. Per Section 2.3.2 Error Conditions, if an error occurs, the digest of the value 00000001h MUST be extended in PCR[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 9.4.1 Event Types.

**Entities that MUST be measured if the TPM is disabled:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call.

**Entities that MUST be measured if the TPM device is hidden:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] prior to the first invocation of the first Ready to Boot call. Platform Firmware SHALL NOT place an event in the event log.

**Entities to exclude:**

1. Portions of the Boot Manager 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, Platform Firmware MUST NOT record the EV_EFI_ACTION event “Calling EFI Application from Boot Option” per Section 9.4.4 EV_EFI_ACTION Event Types when attempting to boot a device.

**Method for measurement:**

This section provides only a short overview of the actions required. See Section 7.2.4 Measuring OS Boot Events for specific details.

The measuring entity MUST measure normalized code for all UEFI applications into PCR[4]. Across boot cycles to the same boot device(s) connected in the same way, Platform Firmware measurements into PCR[4] MUST be measured in the same sequence every time.

Platform Firmware performs these steps as follows:

1. Measure EV_OMIT_BOOT_DEVICE_EVENTS if Platform Firmware is designed or configured to not record which devices it attempts to boot.
2. Measure EV_ACTION event “Calling Ready to Boot”. See Section 9.4.3 EV_ACTION Event Types.
3. Measure EV_SEPARATOR in PCR[0-7]. See Section 9.4.1 Event Types.
4. Measure EV_ACTION event “Returned Ready to Boot”. See Section 9.4.3 EV_ACTION Event Types.
5. Measure EV_EFI_ACTION “Calling EFI Application from Boot Option” and optionally 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. See Section 9.4.4 EV_EFI_ACTION Event Types and Section 9.4.3 EV_ACTION Event Types.
6. Measure the Boot Manager Code before transferring control to it. (Note: Some implementations may load Boot Manager Code, but decide not to execute it. An example is Platform Firmware designed to read from a USB device to determine if it is bootable and conditionally run code from the device if it is. If the USB device is not bootable, Platform Firmware does not measure it because it never transfers control to it.)
7. Conditionally measure the Boot Manager configuration data in PCR[5] if appropriate for the class of device that is booted per Section 7.2.4 Measuring OS Boot Events.
8. Boot Manager Code SHOULD measure additional Boot Manager Code to which it transfers control into PCR[4] or into PCRs reserved for the OS.
9. Boot Manager Code SHOULD measure additional configuration data into PCR[5] or into PCRs reserved for the OS.

10. If control returns to Platform Firmware, measure the returning event as EV_ACTION event “Returned via INT 18h” OR “Returned Ready to Boot”, depending on how the device returns control to Platform Firmware. See Section 9.4.3 EV_ACTION Event Types.
11. For additional boot devices or paths, go to Step 5.

#### 2.3.4.6 PCR[5] – Boot Manager Configuration and Data

**Start of informative comment**

The Boot Manager 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, Platform Firmware will record the Boot Manager Data or configuration information. In other cases, Boot Manager Code or UEFI Application 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 Platform Firmware is security-relevant data associated with booting the device. An example is data embedded within the Boot Manager Code such as the disk geometry within the GPT.

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

Another example of UEFI Application recording configuration data is PXE code measuring partition data it loaded using the EV_EFI_ACTION event and encoding the data as an ASCII string.

This section contains the PCR[5] measurement requirements for a UEFI platform.

In general, entities measure into PCR[5] data associated with the code modules measured into PCR[4]. For a UEFI platform, this includes but is not limited to

- The GPT/Partition Table (as shown in Figure 6 PCR Mapping of UEFI Components)

Additional variables that may impact system behavior beyond the boot options that are measured into PCR[1] (the source of these additional variables may be the UEFI Specification or private variables) may be optionally measured by the UEFI boot application. These loader variables shall be measured into PCR[5]. The source of these additional variables may be the UEFI Specification or private variables. These variables can include, but are not limited to, language code, and so on.

**End of informative comment**

**Entities that MUST be measured if the TPM is enabled:**

1. All relevant Boot Manager configuration data, using the event EV_EFI_VARIABLE_DRIVER_CONFIG. See Section 9.4.1 Event Types.
2. Security-relevant data contained within the Boot Manager Code (e.g., disk geometry), using the event EV_EFI_GPT_EVENT. See Section 9.4.4 EV_EFI_ACTION Event Types for what to place in the event data field.
3. If applicable, when booting a UEFI Application or an adaptor that hosts an Application, it MUST measure other data, including comments, specific to the device using the event type EV_EFI_VARIABLE_DRIVER_CONFIG with an ASCII string “<vendor specific string>” value as described in Section 9.4.4 EV_EFI_ACTION Event Types as determined by the device manufacturer.
4. Platform Firmware MUST record the EV_EFI_ACTION event “Exit Boot Services Invocation”. See Section 9.4.4 EV_EFI_ACTION Event Types.

5. Per Section 7.2.4 Measuring OS Boot Events, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call. See Section 9.4.1 Event Types.
6. Per Section 2.3.2 Error Conditions, if an error occurs, the digest of the value 00000001h MUST be extended in PCR[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 9.4.1 Event Types.

**Entities that MAY be measured if the TPM is enabled:**

1. The GPT Table MAY be measured using the event type EV_EFI_GPT_EVENT.
2. EFI variables, either defined in the UEFI specification or private, that typically do not change from boot to boot and contain system configuration information MAY be measured using EV_EFI_VARIABLE_DRIVER_CONFIG.

**Entities that MUST be measured if the TPM is disabled:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call.

**Entities that MUST be measured if the TPM device is hidden:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] prior to the first invocation of the first Ready to Boot call. Platform Firmware SHALL NOT place an event in the event log.

**Method for measurement:**

1. The EV_SEPARATOR event prior to Ready to Boot invocation.
2. Platform Firmware measures the static data such as disk geometry.
3. The Boot Manager Code measures all relevant IPL configuration data per its defined events.

#### 2.3.4.7 PCR[6] – Host Platform Manufacturer Specific Measurements

**Start of informative comment**

This PCR is reserved for use by the Host Platform manufacturer. This allows for Host Platform Manufacturer-specific 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.

Operating systems 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. A platform manufacturer may have difficulty extending PCR[6] without interfering with OS management of the TPM. In this case, a platform manufacturer would need to work with their OS vendor to implement a mechanism to record measurements in the OS-present event log.

Previously defined measurements for PCR[6] from the PC Client Implementation Specification for Conventional BIOS have been deprecated.

**End of informative comment**

1. The Host Platform Manufacturer MAY define the purpose of this PCR.
2. If the Platform Manufacturer extends PCR[6] during the Pre-OS environment, it MUST record a corresponding log entry.
3. If the Platform Manufacturer extends PCR[6] 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 enabled:**

1. Per Section 7.2.4 Measuring OS Boot Events, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call. See Section 9.4.1 Event Types.
2. Per Section 2.3.2 Error Conditions, if an error occurs, the digest of the value 00000001h MUST be extended in PCR[0-7] and an EV_SEPARATOR event SHOULD be recorded in the event log for each PCR. See Section 9.4.1 Event Types.

**Entities that MAY be measured if the TPM is enabled:**

1. The Host Platform Manufacturer MAY measure platform specific events using the event type EV_COMPACT_HASH. If measured, the Event Data field SHALL be a unique string.

**Entities that MUST be measured if the TPM is disabled:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call.

**Entities that MUST be measured if the TPM device is hidden:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] prior to the first invocation of the first Ready to Boot call. Platform Firmware SHALL NOT place an event in the event log.

**Method for measurement:**

1. The EV_SEPARATOR event prior to Ready to Boot invocation MUST be the last Platform Firmware initiated measurement in the PCR.

#### 2.3.4.8 PCR[7] – Secure Boot Policy Measurements

**Start of informative comment**

This section contains the PCR[7] measurement requirements.

In addition, PCR[7] is used to record secure boot policy information as described below. See Chapter 27 of the UEFI Specification for more information on UEFI Secure Boot.

PCR[7] is used to reflect the UEFI 2.3.1 Secure Boot policy. This policy relies on the firmware authenticating all boot components launched prior to the UEFI environment and the UEFI platform initialization code (or earlier firmware code) invariantly recording the Secure Boot policy information into PCR[7].

**End of informative comment**

Platform Firmware adhering to the policy MUST therefore measure the following values into PCR[7]:

 

1. The contents of the SecureBoot variable
2. The contents of the PK variable
3. The contents of the KEK variable
4. The contents of the UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE variable
5. The contents of the UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE1 variable
6. Entries in the UEFI_IMAGE_SECURITY_DATABASE that are used to validate UEFI Drivers or UEFI Boot Applications in the boot path
7. The contents of the UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE2 7. variable, if present and not empty
8. If the system supports UEFI 2.5 or later, the contents of the UEFI_IMAGE_SECURITY_DATABASE_GUID/EFI_IMAGE_SECURITY_DATABASE3 if present and not empty.
9. If the system supports UEFI 2.5 or later and DeployedMode is NOT enabled, the following additional variables MUST be measured into PCR[7]:
    a. The contents of the AuditMode variable
    b. The contents of the DeployedMode variable

For all UEFI variable value events, the eventType SHALL be EV_EFI_VARIABLE_DRIVER_CONFIG and the Event value SHALL be the value of the UEFI_VARIABLE_DATA structure (this structure SHALL be considered byte-aligned).

The measurement digest MUST be tagged Hash for each supported PCR bank of the event data which is the UEFI_VARIABLE_DATA structure. (Note: This is a different digest than the one used in the previous version of this specification.) The UEFI_VARIABLE_DATA.UnicodeNameLength value is the number of CHAR16 characters (not the number of bytes). The UEFI_VARIABLE_DATA.UnicodeName contents MUST NOT include a NUL.

Images that LoadImage fails to load due to (a) signature verification failure or (b) because the image does not comply with currently enforced UEFI 2.3.1 Secure Boot policy do not need to be measured in a PCR.

If reading a UEFI variable returns UEFI_NOT_FOUND, the UEFI_VARIABLE_DATA.VariableDataLength field MUST be set to zero and UEFI_VARIABLE_DATA.VariableData field will have a size of zero.

**Entities that MUST be measured if the TPM is enabled:**

1. If the platform provides a firmware debugger mode which may be used prior to the UEFI environment or if the platform provides a debugger for the UEFI environment, then the platform SHALL extend an EV_EFI_ACTION event into PCR[7] before allowing use of the debugger. The event string SHALL be “UEFI Debug Mode”. The Platform Firmware MUST log this measurement in the event log using the string “UEFI Debug Mode” for the Event Data.
2. Before executing any code not cryptographically authenticated as being provided by the Platform Manufacturer, the Platform Manufacturer firmware MUST measure the following values, in the order listed using the EV_EFI_VARIABLE_DRIVER_CONFIG event type to PCR[7]:
    a. SecureBoot variable value
    b. The PK variable value
    c. The KEK variable value
    d. The UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE variable value
    e. The UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE1 variable value

f. The UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE2 variable value, if present and not empty
g. If the system supports UEFI 2.5 or later, the UEFI_IMAGE_SECURITY_DATABASE_GUID/EFI_IMAGE_SECURITY_DATABASE3 variable value if present and not empty
h. If the system supports UEFI 2.5 or later and DeployedMode is NOT enabled, the AuditMode and DeployedMode variable values if present and not empty

3. If the platform supports changing any of the following UEFI policy variables after they are initially measured in PCR[7] and before ExitBootServices() has completed and the platform does not support UEFI 2.5 or later, the platform MUST be restarted. If the platform supports UEFI 2.5 or later and supports changing any of the following UEFI policy variables after they are initially measured in PCR[7] and before ExitBootServices() has completed, the platform MAY be restarted OR the variables MUST be remeasured into PCR[7]. Additionally the normal update process for setting any of the UEFI variables below SHOULD occur before the initial measurement in PCR[7] or after the call to ExitBootServices() has completed.
    a. SecureBoot variable value
    b. The PK variable value
    c. The KEK variable value
    d. The UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE variable value
    e. The UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE1 variable value
    f. The UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE2 variable value if present and not empty
    g. If the system supports UEFI 2.5 or later, the UEFI_IMAGE_SECURITY_DATABASE_GUID/EFI_IMAGE_SECURITY_DATABASE3 variable value
    h. If the system supports UEFI 2.5 or later, DeployedMode variable value
    i. If the system supports UEFI 2.5 or later, the AuditMode variable value.
4. The system SHALL measure the EV_SEPARATOR event in PCR[7]. (This occurs at the same time the separator is measured to PCR[0-7].)

   Before launching a UEFI Driver or a UEFI Boot Application (and regardless of whether the launch is due to the UEFI Boot Manager picking an image from the DriverOrder or BootOrder UEFI variables or an already launched image calling the UEFI LoadImage() function), the UEFI firmware SHALL determine if the entry in the UEFI_IMAGE_SECURITY_DATABASE_GUID/ EFI_IMAGE_SECURITY_DATABASE variable that was used to validate the UEFI image has previously been measured in PCR[7]. If it has not been, it MUST be measured into PCR[7] as follows. If it has been measured previously, it MUST NOT be measured again. The measurement SHALL occur in conjunction with image load.
    a. The eventType SHALL be EV_EFI_VARIABLE_AUTHORITY.
    b. The event value SHALL be the value of the UEFI_VARIABLE_DATA structure.
        i. The UEFI_VARIABLE_DATA.VariableData value SHALL be the UEFI_SIGNATURE_DATA value from the UEFI_SIGNATURE_LIST that contained the authority that was used to validate the image.
        ii. The UEFI_VARIABLE_DATA.VariableName SHALL be set to UEFI_IMAGE_SECURITY_DATABASE_GUID.
        iii. The UEFI_VARIABLE_DATA.UnicodeName SHALL be set to the value of UEFI_IMAGE_SECURITY_DATABASE. The value MUST NOT include the terminating NUL.
5. The EV_EFI_VARIABLE_AUTHORITY measurement in step 6 is not required if the value of the SecureBoot variable is 00h (off). In such cases, no validation occurs against the SecureBoot databases.

**Entities that MUST be measured if the TPM is disabled:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] and an EV_SEPARATOR event MUST be recorded in the event log for PCR[0-7] prior to the first invocation of the first Ready to Boot call.

**Entities that MUST be measured if the TPM device is hidden:**

1. Per Section 2.3.1 Collection and Reporting of Measurements, the digest of the value 00000000h or FFFFFFFFh MUST be extended in PCR[0-7] prior to the first invocation of the first Ready to Boot call. Platform Firmware SHALL NOT place an event in the event log.

#### 2.3.4.9 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., TPM2_Create). 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. 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.

#### 2.3.4.10 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**

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

### 2.3.5 Localities assigned to RTM's

#### 2.3.5.1 H-CRTM Localities and Concepts

**Start of informative comment**

The H-CRTM sequence is initiated at the earliest stage of platform boot by the CPU. It is used to extend one or more events to PCR[0]. The H-CRTM sequence uses the interface commands _TPM_Hash_Start, _TPM_Hash_Data, and _TPM_Hash_End, prior to the TPM receiving a TPM2_Startup command. The DRTM sequence uses _TPM_Hash_Start, _TPM_Hash_Data, and _TPM_Hash_End after the TPM has received the TPM2_Startup command. When an H-CRTM sequence occurs, the Platform Firmware has not started, and will not know that one occurred. Because of this, it is required that the code initiating an H-CRTM sequence leave some artifacts for Platform Firmware to construct the events it will record in the event log on behalf of the H-CRTM code.

 

When an H-CRTM sequence occurs, the code initiating the sequence will also send the TPM2_Startup command. To avoid sending TPM2_Startup twice, an additional artifact will be created for Platform Firmware to understand the state of the TPM when it gets control of the boot process. Given that the TPM2_Startup can occur from locality 3, if an H-CRTM sequence occurs, Platform Firmware will construct the Startup Locality Event (See Section 9.4.5.3 Startup Locality Event).

**End of informative comment**

#### 2.3.4.2 DRTM and Transitive Trust Chain

**Start of informative comment**

The Host Platform Manufacturer and Dynamic OS define the usage of Localities 1–4 and PCRs 17–22. While 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 Platform TPM Profile for TPM 2.0. For more information please see the TCG D-RTM Architecture Specification Version 1.0.

**End of informative comment**

#### 2.3.5.2.1 Localities 1-3

##### 2.3.5.2.1.1 Integrity Collection 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 DRTM.

##### 2.3.5.2.1.2 PCR Usage

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

##### 2.3.5.2.1.3 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.

#### 2.3.5.2.2 Locality 4

##### 2.3.5.2.2.1 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 TPM2_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 DRTM.

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 DRTM.

#### 2.3.5.2.2.2 Integrity Collection and Reporting

How this is done is Host Platform implementation-specific.

#### 2.3.5.2.2.3 PCR Usage

The PCR associated with this Locality is PCR[17]. This Locality MUST measure the first component executed after the DRTM and MAY measure other components as determined by Host Platform's implementation. This is analogous to PCR[0] in the SRTM.

#### 2.3.5.2.2.4 Protection

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

# 3 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 is required to be provided by the TPM.

**End of informative comment**

## 3.1 NV RAM Size and Allocation

**Start of informative comment**

The PC Client Platform TPM Profile for TPM 2.0 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 were written.

**End of informative comment**

# 4 Maintenance

**Start of informative comment**

Maintenance for the PC Specific Implementation Specification refers to the processes surrounding upgrade or replacement of the Platform Firmware ROM / Flash. All requirements for TPM maintenance are manufacturer-defined per the TPM Library Specification for Family 2.0.

**End of informative comment**

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

1. Firmware meeting the requirements of the US National Institute of Standards and Technology (NIST) Special Publication 800-155 BIOS Integrity Measurement Guidelines SHALL produce one or more identifiers used to associate the platform with its reference manifest (RM). These identifiers are EV_NO_ACTION log events containing an event of type TCG_Sp800-155-PlatformId_Event. See Section 9.4.5.2 Firmware Integrity Measurement Reference Manifest Event.

## 4.1 Platform Firmware Recovery Mode

**Start of informative comment**

This is a failure-recovery mode of Platform Firmware that is invoked by the Boot Block typically when the UEFI Firmware is corrupt. The Platform Firmware Recovery Mode will perform a minimal initialization of the system and then attempt to boot a recovery program from some type of external media. The Platform Firmware Recovery Mode may not have the capability to continue the SRTM measurement chain.

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

1. Use of hardware to disable the TPM until the next _TPM_Init; or,
2. Disabling the TPM by calling TPM2_Startup (CLEAR) followed by TPM2_HierarchyControl (EH Disable, SH Disable); or,
3. Measuring of components to maintain the SRTM for BIOS Recovery Mode components; or,

Unconditionally performing a Host Platform Reset upon completion of BIOS Recovery Code.

**End of informative comment**

1. It MUST NOT be possible for a BIOS Recovery Mode to allow impersonation of another boot state as represented by the SRTM. This applies to the values in the PCR[0-7].
2. If Recovery requires a boot in Recovery Mode, the Firmware SHALL:
    a. Initialize the TPM in a disabled state by sending a TPM2_Startup (CLEAR) followed by a TPM2_HierarchyControl (EH disable, SH disable) to disable each of the hierarchies.
    b. Cap PCR[0-7] in all active PCR banks with the digest of the value 00000000h or FFFFFFFFh and record an EV_SEPARATOR Event in the event log. See Section 9.4.1 Event Types.

## 4.2 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 Platform Firmware 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.
2. The MAE is vendor-specific and MUST provide protections for the TBB and SRTM.
3. At the completion of the update process, the reset vector MUST point to the one and only one SRTM that is controlled by the MAE.
4. At the completion of the update process, the execution MUST begin at the SRTM designated by the platform manufacturer.
5. If a BIOS update is installed that modifies the SRTM, Platform Firmware update process MUST unconditionally transition the platform to the Off state or reboot.

## 4.3 Firmware Compliance Requirements

The UEFI Firmware SHOULD meet the requirements of the U.S. National Institute of Standards and Technology *(NIST) Special Publication 800-147 Bios Protection Guidelines* available at: http://csrc.nist.gov/publications/nistpubs/800-147/NIST-SP800-147-April2011.pdf.

# 5 TCG Certificates and Verification of a Platform for SP800-155 Compliance

**Start of informative comment**

The UEFI Specification describes a methodology for providing credentials in a PE/COFF image.

This section describes the obligations of a platform that supports NIST SP 800-155 BIOS Integrity Measurement Guidelines. In addition to defining obligations of the different Roots of Trust (RTR, RTM, RTS), 800-155 defines two additional entities: the Measurement Assessment Authority (MAA) and a Reference Manifest (RM). Both the MAA and the RM are outside of the UEFI firmware, but the UEFI firmware needs to provide some indication so that the MAA can ascertain the appropriate RM for the platform. The TCG Firmware Integrity Measurement (FIM) document defines a Verifier that fulfils the role of the MAA and Reference Integrity Manifest (RIM) that provides "Golden Measurements" to the Verifier.

An event log contains one or more TCG_Sp800-155-PlatformId_Event2 structures that the Verifier uses to identify the corresponding vendor and RIM that contains the same VendorId and ReferenceManifestGuid components as included in the TCG_Sp800-155-PlatformId_Event2 structure. The Verifier obtains RIM(s) by a mechanism outside the scope of this specification. When the event log contains multiple TCG_Sp800-155-PlatformId_Event2 structures then multiple RIMs are needed, where each RIM contains different sets of golden measurements.

The following example is for illustrative purposes only and in no way implies a recommended implementation.

For a platform's flash containing the following firmware volumes (FV):

PEI FV (CRTM) – c1

DXE FV (Main compressed BIOS – c2

OEM DXE Extension FV – c3

Variable Block

OEM Vital Product Data (VPD)

For a boot sequence where FV c1 and c2 executed, the corresponding event log could contain the following events:

Specification ID Version Event (See Section 9.4.5.1 Specification ID Version Event.)

TCG_Sp800-155-PlatformId_Event2 1

TCG_Sp800-155-PlatformId_Event2 2

PCR[0]: CRTM Version Event – hash of c1's version

PCR[0]: DXE FV Event – hash of c2

PCR[1]: events

PCR[2]: events

Etc.

In this example: TCG_Sp800-155-PlatformId_Event2 1 could identify the RIM containing the golden measurement for c1 while TCG_Sp800-155-PlatformId_Event2 2 could identify the RIM containing the golden measurement for c2.

**End of informative comment**

 

The UEFI Firmware SHOULD meet the requirements of the U.S. National Institute of Standards and Technology (NIST) Special Publication 800-155 BIOS Integrity Measurement Guidelines available at: http://csrc.nist.gov/publications/drafts/800-155/draft-SP800-155_Dec2011.pdf.

For the NIST 800-155 platform attributes (i.e.: hardware protection of the flash, or 800-147 support), if static, this should be in the platform's Platform Certificate. If not static, then an optional log entry into PCR[1], which could contain those characteristics using the EV_PLATFORM_CONFIG_FLAGS event should be created.

# 6 TPM Discoverability

**Start of informative comment**

The TPM powers up with all hierarchies enabled and all functions available by default. This is different than TPM 1.2, which followed a gradual opt-in model. There are cases where a Platform OEM may decide to configure a TPM so that it is hidden from an end-user, an operating system, or OS present applications. An example of such a case may be where the TPM is restricted by regulation from being imported into a specific country or geography. There are other cases where an end-user might decide that they want to disable part or all of the TPM to ensure it is not usable by OS present applications.

This section describes the requirements for Platform OEMs when they provide the ability for an end-user or Platform Firmware to selectively or completely disable the TPM. Additional requirements may be detailed in other sections.

The PC Client Implementation Specification for Conventional BIOS used the terms “enumeration” and “discoverable” to describe these states of the TPM where the TPM was visible or not to the OS and the end-user. These terms were generally confusing and so are not used in this specification. The requirements are defined in relation to visibility of the TPM to the OS present interface and to the end-user through BIOS Setup.

**End of informative comment**

## 6.1 TPM Visibility to the OS

**Start of informative comment**

The default state of the TPM makes it ready and available to the OS and OS present applications.

**Note:** Disabling the platform hierarchy will hide all NV Indices in the TPM with the attributes platformCreate SET and phEnableNV CLEAR. Hiding the TPM, but leaving the platform hierarchy enabled may allow a dictionary attack to force the TPM into Lockout.

**End of informative comment**

If the Platform OEM implements a capability to prevent the OS from discovering the TPM, or hiding it from the OS and OS-present applications, the firmware SHALL implement all of the following requirements:

1. When the TPM is visible to the OS and enabled:
    a. The TPM2 ACPI Table SHALL be listed in the RSDT and XSDT tables as described in Section 8.1 ACPI Device Object for TPM.
    b. The TPM device object SHALL be present in the ACPI namespace. See 8.1 ACPI Device Object for TPM and STA() shall return 0xF.
    c. The Platform OEM SHALL make all measurements in compliance with the requirements of this specification.
    d. The Platform OEM-provided mechanism to make the TPM hidden from the OS SHALL perform a Host Platform Reset.
    e. The TPM SHALL remain visible to the OS on resume from sleep or hibernate.
    f. The Firmware SHALL support the EFI_TCG2_PROTOCOL as specified in the TCG EFI Protocol Specification.
    g. The EFI_TCG2_BOOT_SERVICE_CAPABILITY.TPMPresentFlag SHALL return TRUE.
    h. The Firmware SHALL support the Physical Presence Interface as specified in the TCG Physical Presence Interface Specification for TPM Family 1.2 and 2.0 revision 1.3 version 52, or later.
    i. The TPM control surface as defined in Section 6.3 Platform Firmware Setup TPM Control Surface SHALL be present in BIOS Setup.
    j. All TPM Hierarchies MUST be enabled, without separate user action to disable one or more hierarchies.
    k. Platform Firmware SHALL populate the TCG Event Log, see Section 9.1 Introduction.

 

l. The TPM state shall not change on resume from sleep or hibernate.

2. When the TPM is visible to the OS but disabled:
   a. The TPM2 ACPI Table SHALL be listed in the RSDT and XSDT tables.
   b. The TPM device object SHALL be present in the ACPI namespace and STA() SHALL return 0xF.
   c. The EFI_TCG2_PROTOCOL SHALL be present.
   d. The EFI_TCG2_BOOT_SERVICE_CAPABILITY.TPMPresentFlag SHALL return TRUE.
   e. The Platform Firmware SHALL disable the Storage and Endorsement Hierarchies by use of the command TPM2_Hierarchy_Control (ehDisable and shDisable).
   f. Platform Firmware MAY disable the Platform Hierarchy.
   g. Platform Firmware SHALL cap PCR[0-7] as described in Section 2.3.4 PCR Usage.
   h. Platform Firmware SHALL create the TCG Event Log as described in Section 9.1 Introduction.
   i. The TPM state shall not change on resume from sleep or hibernate
3. When the TPM is discoverable but hidden from the OS:
   a. The TPM2 ACPI Table SHALL be listed in RSDT and XDST tables.
   b. The TPM device object SHALL be present in the ACPI namespace and STA() SHALL return 0x0.
   c. The EFI_TCG2_PROTOCOL SHALL be present.
   d. The EFI_TCG_BOOT_SERVICE_CAPABLITY.TPMPresentFlag SHALL return FALSE.
   e. Platform Firmware SHALL disable the Storage and Endorsement Hierarchies by use of the command TPM2_Hierarchy_Control (ehDisable and shDisable).
   f. Platform Firmware MAY disable the Platform Hierarchy.
   g. Platform Firmware SHALL cap PCR[0-7] as described in Section 2.3.4 PCR Usage.
   h. Platform Firmware MAY create the TCG Event Log.
   i. The Platform OEM-provided mechanism to make the TPM visible to the OS SHALL perform a Host Platform Reset.
   j. The TPM state shall not change on resume from sleep or hibernate.
4. When the TPM is not discoverable and is hidden from the OS:
   a. The TPM2 ACPI Table SHALL NOT be listed in the RSDT or XSDT tables.
   b. The TPM device object SHALL NOT be present in the ACPI namespace.
   c. The EFI_TCG2_EFI_PROTOCOL SHALL NOT be present.
   d. The TPM SHALL not be usable by OS-present applications.
   e. The TPM Storage and Endorsement Hierarchies SHALL be disabled by Firmware on every boot using a TPM2_HierarchyControl command for both hierarchies.
   f. The TPM Platform Hierarchy MAY be disabled by Firmware on every boot using a TPM2_HierarchyEnable command.
   g. Firmware SHALL cap PCR[0-7] as described in Section 2.3.4 PCR Usage.
   h. Platform Firmware SHALL NOT create the TCG Event Log.
   i. The Platform OEM-provided mechanism to make the TPM visible to the OS SHALL perform a Host Platform Reset.
   j. The TPM SHALL remain hidden from the OS on resume from sleep or hibernate.

## 6.2 TPM Visibility to End-Users through BIOS Setup

**Start of informative comment**

PC Client OEMs will provide the control surface for TPM through Platform Firmware Setup menu. In the event that the TPM is completely hidden and disabled, the user interface for TPM will also be absent from BIOS Setup.

**Note:** Disabling the platform hierarchy will hide all NV Indices in the TPM with the attributes platformCreate SET and phEnableNV CLEAR. Hiding the TPM, but leaving the platform hierarchy enabled may allow a dictionary attack to force the TPM into Lockout.

**End of informative comment**

If the Platform OEM implements a capability to hide the TPM from the end-user, the firmware SHALL implement all of the requirements in Section 6.1 TPM Visibility to the OS normative 4 and the following additional requirements:

1. The TPM control surface as defined in Section 6.3 Platform Firmware Setup TPM Control Surface SHALL NOT be present in BIOS Setup.

## 6.3 Platform Firmware Setup TPM Control Surface

**Start of informative comment**

PC OEMs present a view of the TPM setup to end users through their Setup utilities. Platforms with TPM 1.2 generally exposed settings only related to enumerating the TPM and modifying the TPM state by enabling, activating or clearing the TPM. Platforms implemented to this specification may continue to support TPM 1.2 in addition to TPM 2.0, e.g. by putting two different TPM devices in the platform or by utilizing a TPM that supports both TPM 1.2 and TPM 2.0 via a firmware update or proprietary switch, but the control surfaces are mutually exclusive. There are many ways a Platform OEM may choose to support both TPM versions and, while this is largely out of scope of this specification, it is important that only one TPM be usable and visible to an end-user and OS on any given boot. This specification does not modify the control surface for TPM 1.2, which is specified in the TCG PC Client Implementation Specification for Conventional BIOS 1.21.

**End of informative comment**

**Table 2 Example Comparison of TPM Family 1.2 and 2.0 Firmware User Interface**

| User Interface Menu Labels for TPM 1.2 | User Interface Menu Labels for TPM 2.0 | User Interface Help Text |
|---|---|---|
| "On/Off" | "On/Off" | TPM 1.2:<br>"On" – TPM enumerated to OS, TPM device object present in ACPI, PPI active<br>"Off" – TPM not enumerated to OS, TPM device object absent from ACPI, PPI optionally inactive;<br>TPM 2.0:<br>"On" – TPM fully operational<br>"Off" – See Section 6.1 TPM Visibility to the OS<br>**NOTE:** This state is OEM specific, User Interface is not defined by TCG only TPM and Event Log behavior is defined by TCG. |
| "Enabled/Disabled" | Enabled/Disabled" | TPM 1.2:<br>"Enabled" – TPM_PhysicalEnable command sent to TPM from FW, TPM enumerated to OS, TPM device object present in ACPI, PPI active<br>"Disabled" – TPM_PhysicalDisable command sent to TPM from FW, TPM enumerated to |

 

| | | |
|---|---|---|
| | | OS, TPM device object present in ACPI, PPI active; default state defined by TCG<br><br>TPM 2.0:<br><br>“Enabled”<br><br>Default state of the TPM, defined by TCG.<br><br>“Disabled”<br><br>See Section 6.1 TPM Visibility to the OS |
| “Activated/Deactivated | n/a | TPM 1.2:<br><br>“Activated” – TPM_PhysicalDeactivate (Activate) command sent to TPM from FW, TPM must be in Enabled state.<br><br>“Deactivated” – TPM_PhysicalDeactivate (Deactivate) command sent to TPM from FW, TPM MAY be Enabled. Default state defined by TCG.<br><br>TPM 2.0:<br><br>These states have no meaning in TPM 2.0. |
| “Clear” | “Clear” | TPM 1.2:<br><br>“Clear”<br><br>TPM_ForceClear command sent to TPM by FW. Requires physical presence. Transitions TPM to Disabled/Deactivated state.<br><br>TPM 2.0:<br><br>“Clear”<br><br>TPM2_Clear command sent to TPM by FW. Requires Platform Authorization or Physical Presence. Clears Storage Hierarchy and EH Proof. TPM remains Enabled. |
| n/a | Measurement Hash Algorithm Selection | TPM 1.2:<br><br>Not supported by TPM 1.2<br><br>TPM 2.0<br><br><Algorithm Selection><br><br>TPM2_PCR_Allocate command sent to the TPM by FW. Requires Platform Authorization or Physical Presence. Sets the Hash |

| | | |
|---|---|---|
| | | Algorithm of the specified PCR bank to the selected algorithm. Requires a reboot. |

1. If the Platform supports both TPM 1.2 and TPM 2.0 family devices:
    a. The Platform Firmware SHALL enable the functionality to support only one TPM family per boot.
    b. The Platform MAY provide an OEM specific mechanism to change support for TPM families.
    c. Platform Firmware SHALL unconditionally reset the platform after changing the TPM family support.
2. If the Platform Firmware supports the ability to disable the Storage and Endorsement Hierarchies:
    a. The Platform Firmware SHALL support a mechanism to enable and disable them in Setup
    b. The Platform Firmware MAY support the Physical Presence Interface mechanism to enable and disable them.
    c. Platform Firmware MAY support a mechanism to enable/disable both hierarchies independently.
    d. Platform Firmware SHALL provide descriptive text describing the implications of disabling the hierarchies.
3. Platform Firmware SHALL provide a mechanism to Clear the TPM from Setup.
4. If the Platform supports changing the algorithm used for measuring events, Platform Firmware MAY support a mechanism to do so through Setup.

# 7 State Transitions

## 7.1 Architecture and Definitions

**Start of informative comment**

A handoff to an operating system generally occurs after Platform Firmware has completed its initialization and testing of the Host Platform hardware. As defined in Section 1.2.11 Boot State Transition, the event that marks the transition from the Pre-OS state to the OS-Present state on the Host Platform is the first invocation of Ready to Boot or an equivalent event.

As stated in Section 3.1 of the UEFI Specification, by the time the Boot Manager code gets invoked, the UEFI firmware has found and initialized all the bootable devices on the Host Platform for this boot cycle and has built a priority-ordered list of those devices, created as Boot### and BootVariable### UEFI variables. Each entry in the list includes a pointer to a UEFI application that is capable of booting an operating system from that device.

The Boot Manager starts at the top of the prioritized list and simply attempts to boot a UEFI application from each bootable device, in turn, by loading and executing the application on that device.

If the UEFI application on a device fails to boot an operating system, it returns control to Boot Manager.

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

TCG-enabled firmware must measure and log the events described in the previous section of this Informative comment. LoadImage() must, without exception, measure into PCR[4] the event of loading and executing UEFI application code from a bootable device; this measurement will automatically result in the proper entry being made into the Event Log. Platform Firmware must not hash any data areas when it measures this event. If secure boot is enabled, LoadImage() will measure the authority from UEFI_IMAGE_SECURITY_DATABASE_GUID / EFI_IMAGE_SECURITY_DATABASE variable that was used to verify the OS Loader into PCR[7]. See Section 2.3.4.8 PCR[7] – Secure Boot Policy Measurements.

If the UEFI application on a bootable device returns back to the Boot manager, that event must be measured.

And, as stated in Section 2.3.4.5 PCR[4] – Boot Manager Code and Boot Attempts of this specification, if Boot Manager Code returns control back to Platform Firmware, each subsequent execution of Boot Manager Code must be separately measured.

**End of informative comment**

## 7.2 Procedure for Pre-OS to OS-Present Transition

**Start of informative comment**

In order to measure the transition from the Pre-OS state to the OS-Present state, a number of steps need to be performed. This section of the specification will outline and describe these steps.

Note that the term ‘EFI OS Loader Load’ is a label for a behavior, not necessarily a label for a code module.

Measure code that boots an operating system on a UEFI platform into PCR[4] and measure data associated with that code into PCR[5].

**End of informative comment**

### 7.2.1 Extending PCR[4]

**Start of informative comment**

 

Just before passing control of the Host Platform to the operating system, the UEFI Firmware needs to perform several actions in order to assure that the chain of measurements of the Host Platform boot process is contiguous. PCR[4] will contain the measurements describing the code which attempts to boot an OS.

**End of informative comment**

### 7.2.2 Extending PCR[5]

**Start of informative comment**

PCR[5] is reserved for any configuration data associated with code modules measured into PCR[4]. Information such as GPT/Partition tables is measured into PCR[5]. PCR[5] may be utilized and extended by any boot loader for variable data.

**End of informative comment**

### 7.2.3 Extending PCR[7]

**Start of informative comment**

Measurements in PCR[7] reflect the UEFI Secure Boot policy, introduced in UEFI 2.3.1.

**End of informative comment**

### 7.2.4 Measuring OS Boot Events

**Start of informative comment**

An Event Log enables a challenger to determine the state of trust of the UEFI platform and enables software on the UEFI platform to reconstruct boot events. The Operating System handoff code needs to fill the Event Log with information about the boot devices used to get to the Operating System. The UEFI platform functions designed for computing hash values and extending PCRs should automatically log the extended events.

However, there are events that must be added to the Event Log that are not the result of a PCR extend operation.

The purpose of this section is to specify all the required events added to the Event Log for OS boot, including events that do not result from a PCR extend operation.

**End of informative comment**

1. The Platform Firmware MUST measure the event EV_SEPARATOR into PCR[0-7] once for each platform boot cycle and the measurement of that event MUST draw the line between leaving the pre-OS environment and entering the OS-Present environment. The data within the event field of the EV_SEPARATOR event MUST be 32 bits (a double-word) of 0's or FF's (that is, 00000000h or FFFFFFFFh) unless an error condition occurs. See Section 2.3.2.2 Errors Recording Measurements.
2. The OS Loader Load code MUST measure, into PCR[4], every attempt to load and execute a OS Loader (a UEFI application).
3. A boot device has a UEFI application. The boot variable describes the location of the application. The Platform Firmware launches the application. If an OS or application that was launched by the Boot Option returns back to UEFI firmware, this affects the transitive trust chain and the return action MUST be measured into PCR[4].
4. Sequence of measuring OS boot events MUST proceed as specified below.
    a. Upon selecting a boot device, the UEFI firmware measures the event type EV_EFI_ACTION "Calling EFI Application from Boot Option" into PCR[4].
    b. Measure EV_SEPARATOR into PCR[0-7]. This occurs only once in the flow.
    c. If UEFI Secure Boot is enabled, measure the entry in the UEFI_IMAGE_SECURITY_DATABASE_GUID /EFI_IMAGE_SECURITY_DATABASE that was used to validate the UEFI image into PCR[7] as described in section 2.3.4.8 PCR[7] – Secure Boot Policy Measurements steps 5 and 6.

 

d. In this optional step, measure the GPT with event type EV_EFI_GPT_EVENT and event data set to the UEFI_GPT_DATA structure into PCR[5].
e. Measure the selected UEFI application code PE/COFF image described by the boot variable with event type EV_EFI_BOOT_SERVICES_APPLICATION into PCR[4].
f. Execute UEFI application from boot variable.
    i. In this optional step, if the executing code from step e loads additional applications or drivers prior to successive steps (i.e., return or exit boot services), the loaded applications or drivers MUST be measured into PCR[4].
g. Measure EV_EFI_ACTION event “Returning from EFI Application from Boot Option” into PCR[4].
h. If the firmware needs to select a next boot device, the UEFI firmware MUST jump to step a.
i. If ExitBootServices() is invoked, then an EV_EFI_ACTION event “Exit Boot Services Invocation” MUST be measured. The return value of ExitBootServices() MUST be reflected in a measured event, into PCR[5], as either “Exit Boot Services Returned with Failure” or “Exit Boot Services Returned with Success”, depending upon the return code from the ExitBootServices() call.

### 7.2.5 Passing Control of the TPM from Pre-OS to OS-Present Environments

**Start of informative comment**

Once Platform Firmware has turned control over to an operating system, the Post-Boot environment will load its own set of drivers and code to access the TPM. This could cause a potential conflict since there may be contention between the Pre-OS and OS-Present environments for use and access of the TPM.

TPM runtime services protocol provides for a solution to this problem through Exit Boot Services (EBS). Once the OS-Present environment has loaded its driver support, it will call this function to disable the UEFI pre-OS services. Boot Manager should not return control back to Platform Firmware after calling Exit Boot Services, because Platform Firmware 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 OS Boot Loader continues the measurement of the boot process.

**End of informative comment**

## 7.3 Power States, Transitions, and TPM Initialization

**Start of informative comment**

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 return an error.

**End of informative comment**

1. If the TPM is visible to the OS as described in Section 6.1 TPM Visibility to the OS, the Host Platform MUST:
    a. 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. S0
            2. S1 OR
            3. S2

 

b. 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.)

### 7.3.1 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 SRTM 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 TPM2_Shutdown(STATE) command and restores it.

**End of informative comment**

1. The TBB MUST be the only entity that can change the TPM device power state.
2. The TBB MUST ensure the TPM is only in the D0, D1, or D2 state when the platform is not under the control of the TBB.
3. The OS booted by Platform Firmware SHALL support ACPI.
4. After issuing a TPM2_Shutdown (STATE) command, the OS SHOULD NOT issue TPM commands before transitioning to S3 without issuing another TPM2_Shutdown (STATE) command.

### 7.3.2 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 Firmware can be summarized as:

1. The SRTM prepares the TPM for use when booting or resuming from S3.
2. If the SRTM is modified, the Firmware needs to reboot instead of resuming from S2 or S3.
3. TPM_INIT is only issued by the SRTM during boot or during S3 resume.
4. During boot and resuming from S3, Platform Firmware issues a TPM2_SelfTest command.

**End of informative comment**

### 7.3.3 Off to S0 (Working)

**Start of informative comment**

This transition is from an Off state to the platform power state that supports 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 in either direction between the Legacy and the S0 state.

If the TPM is disabled or the platform owner has disabled measurements, the SRTM may issue the TPM2_Startup and TPM2_HierarchyControl commands such that the TPM is disabled. The SRTM would further cap PCR[0-7] with an EV_SEPARATOR Event. See Section 9.4.1 Event Types.

Transitioning out of S4 is similar to transitioning out of S5 except the FW or OS loader restore a memory image from persistent storage. The pre-OS components are measured normally for the current platform state, including the components that restore the memory image. It is not permitted for the Platform Firmware to process a PPI request following a resume from hibernate without initiating a host platform reset. This is not reflected in Figure 7, which only describes the transition from S5 to S0.

The FW is responsible for issuing the TPM2_SelfTest command, immediately following TPM2_Startup (Clear).

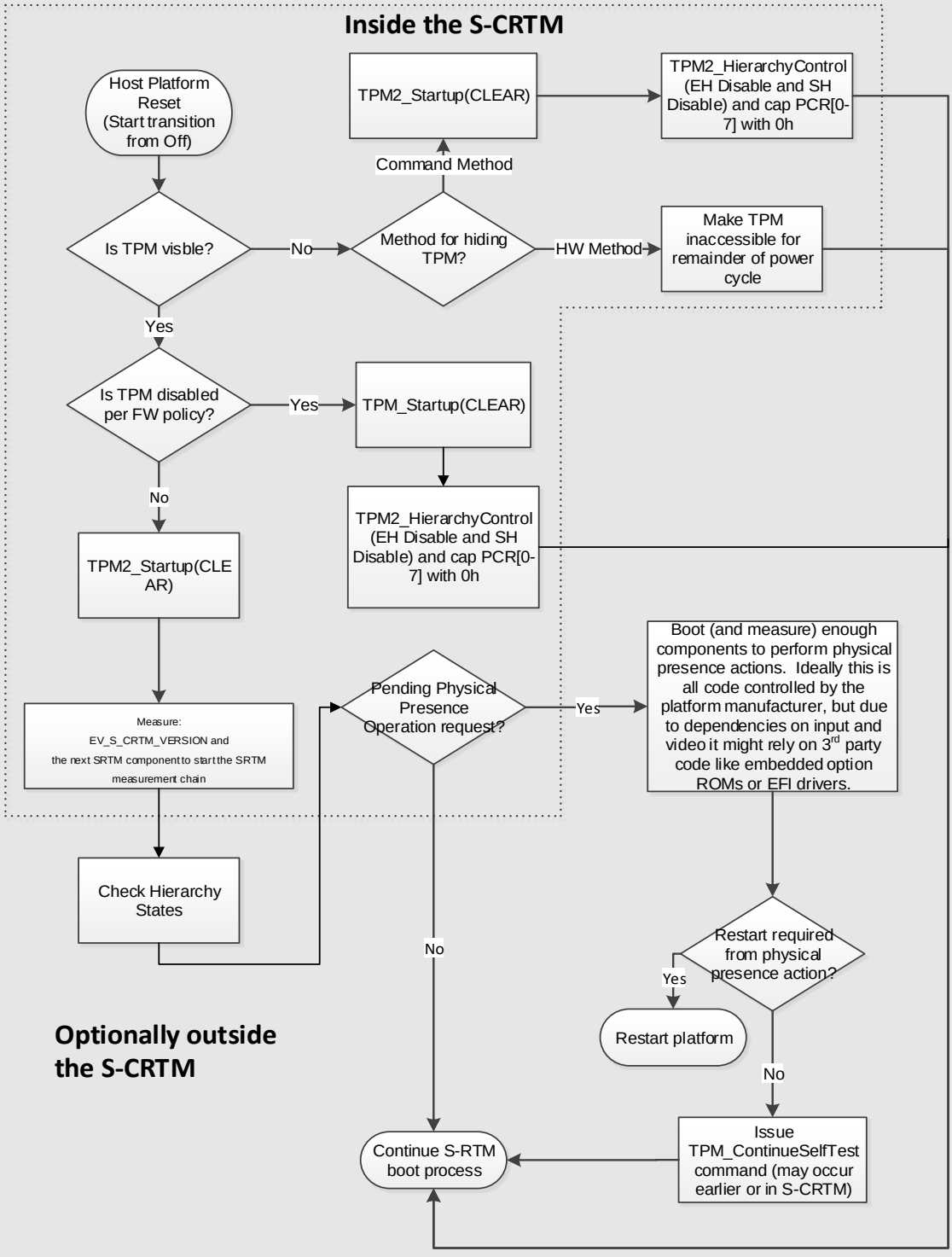


**Figure 7 Firmware Actions during transitions from Off**

**End of informative comment**

When transitioning from the Off state to the S0 states:

1. The Host Platform MUST issue a _TPM_INIT.
2. If the TPM interface is accessible and the TPM is hidden from the OS, the SRTM MUST
   a. Issue a TPM2_Startup (CLEAR) command,
   b. Issue a TPM2_HierarchyControl (EH Disable and SH Disable) command, and
   c. Cap PCR[0-7] by extending in all active PCR banks the digest of the value 00000000h or FFFFFFFFh and recording an EV_SEPARATOR Event in the event log for each PCR, see Section 9.4.1 Event Types.
3. If the TPM interface is accessible and the TPM is enumerated, the SRTM MUST issue a TPM2_SelfTest (FULLTEST = NO) command prior to the first invocation of the first Ready to Boot call.
4. If the TPM is visible to the OS and the TPM interface is accessible, the platform manufacturer MUST set platformAuth to a high entropy value and MAY set platformPolicy during execution of the SRTM such that later software is unable change objects in the Platform Hierarchy or operations that require platform authorization.
5. If the TPM is enabled and visible, the SRTM MUST start the SRTM measurement chain by recording integrity measurements during the boot process per Section 2.3.1 Collection and Reporting of Measurements.
6. Boot Managers SHOULD be prepared for the TPM's Self-Test actions associated with the TPM2_SelfTest command to be in progress when the Boot Manager is launched.

### 7.3.4 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 condition is whether the platform performed an orderly shutdown or unexpectedly lost power. If 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**

### 7.3.5 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.

### 7.3.6 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 FW 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 FW update has occurred since the last transition from S5 to S0, 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 SRTM is responsible for detecting modified FW 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 SRTM measurements in PCR[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 FW update has occurred since the last transition from an Off state to S0.

### 7.3.7 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 7.3.6 S0 (Working) to S2 (Sleep).

Figure 8 Firmware Actions for S2 Resume illustrates the SRTM actions when resuming from S2.

 

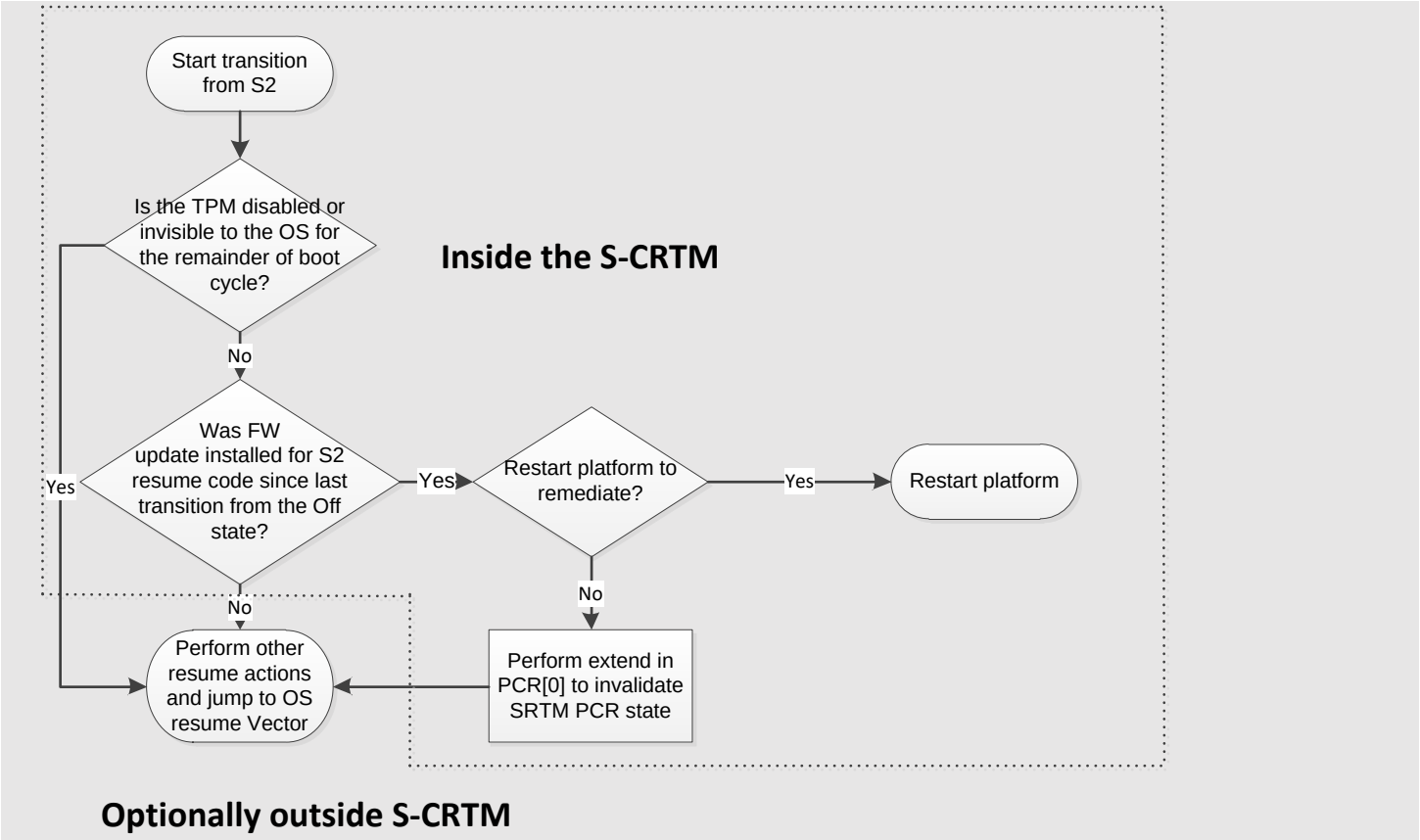


**Figure 8 Firmware Actions for S2 Resume**

**End of informative comment**

1. The Host Platform MUST NOT issue a TPM_INIT.
2. If the TPM is visible and enabled:
   a. The SRTM MUST be able to determine whether there has been an update to any S2 resume portion of the Platform FW since the previous transition from an Off state. The SRTM 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 SRTM to re-measure Platform FW code and compare this measurement to the measurement of FW code performed at the previous transition from S4 or S5. The determination of whether Platform Firmware changed MAY be done using a simple flag.
   b. If the SRTM detects a modification to the S2 resume portion of the Platform Firmware since the last transition from an Off state, the SRTM MUST either:
      i. Force the Host Platform to reboot, or
   c. Make the contents of PCR[0] invalid by extending the digest of the value 00000001h in PCR[0]. Note: No corresponding event is logged in the event log because the OS may be managing the log.

### 7.3.8 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 7.3.6 S0 (Working) to S2 (Sleep) for a special case regarding FW updates.

 

Entering into and exiting from the S3 state is a coordinated effort between the OS and the FW. 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 FW 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 TPM2_Shutdown(STATE) 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 TPM2_Startup command. The initial state of the TPM can only be determined by the components of the RTM. Therefore, the SRTM makes the determination as to whether the Host Platform is resuming from an Off state or the S3 power state. The SRTM must issue the TPM2_Startup command with the appropriate parameter, indicating to the TPM whether to initialize or restore the previously saved state of the TPM.

If TPM2_Startup(STATE) is called when there is no saved state to restore, the TPM returns TPM_RC_VALUE. The SRTM should perform a host platform reset and send a TPM2_Startup(CLEAR) before passing control to the Operating System

**Note:** See the requirement in Section 7.3.1 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 TPM2_Shutdown(STATE) 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 Firmware update has occurred since the last transition from an Off state to S0 or to the Legacy state.

### 7.3.9 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 SRTM issues the TPM2_Startup(STATE) command, loading the previously saved state, without re-measuring Pre-OS components. The SRTM 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 7.3.8 S0 (Working) to S3 (Sleep) for more informative text regarding this transition.

The Platform Firmware 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.

Figure 9 Firmware Actions for Resume from S3 below illustrates the logic for the SRTM actions when resuming from S3.

 

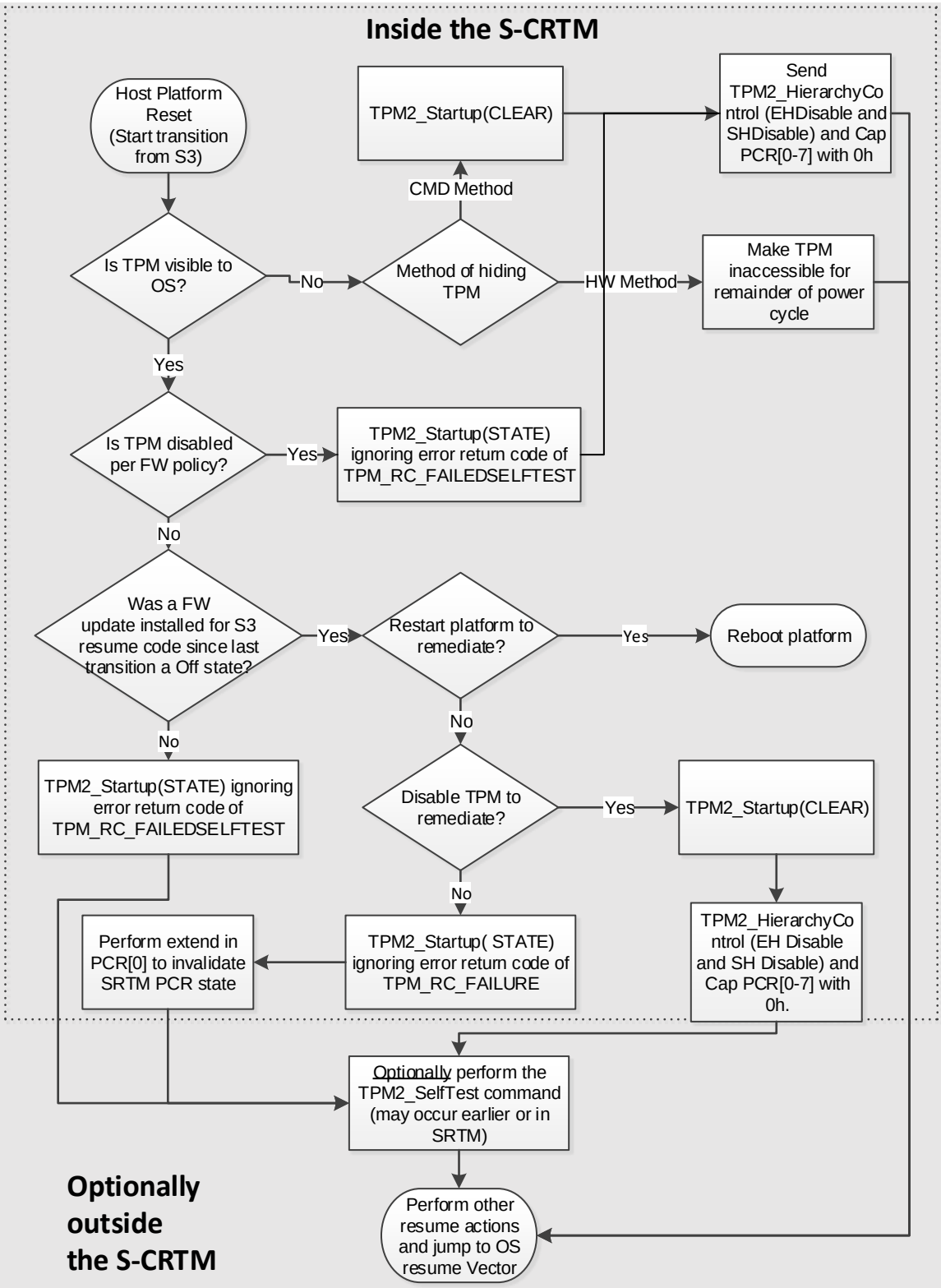


**Figure 9 Firmware Actions for Resume from S3**

 

**Note:** In Figure 9 Firmware Actions for Resume from S3, if the TPM2_Startup(STATE) returns TPM_RC_FAILURE, it isn't necessary to extend an error in PCR[0] to invalidate the SRTM PCR state because the TPM is in failure mode

**End of informative comment**

1. The Host Platform MUST issue a TPM_INIT.
2. The SRTM MUST issue a TPM2_Startup command.
    a. Under these conditions the SRTM MAY issue a TPM2_Startup(CLEAR) command if:
        i. The TPM is hidden from the OS,
        ii. The TPM is disabled per FW policy, or
        iii. A FW update has occurred for the S2 or S3 portion of the Platform FW since the last transition from S5.
    b. Otherwise, the SRTM MUST issue the TPM2_Startup(STATE) command.
    c. When issuing the TPM2_Startup(STATE), the SRTM SHOULD ignore an error resulting from the TPM entering failure mode (TPM_RC_FAILURE).
3. If the TPM is visible and enabled:
    a. The SRTM MUST be able to determine whether there has been an update to any S3 resume portion of the FW since the previous transition from an Off state. **Note:** The SRTM 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 SRTM to re-measure FW and compare this measurement to the measurement of FW performed at the previous transition from an Off state. The determination of whether FW changed MAY be done using a simple flag.
    b. If the SRTM detects a modification to the S3 resume portion of the FW since the last transition from an Off state, the SRTM MUST either:
        i. Force the Host Platform to reboot, or
        ii. The SRTM MUST:
            1. Issue a TPM2_Startup(CLEAR) command,
            2. Issue a TPM2_HierarchyControl (EH Disable, SH Disable) command, and
            3. Make the contents of PCR[0] invalid by extending the digest of the value 00000001h in PCR[0]. Note: No corresponding event is logged in the event log because the OS may be managing the log.
4. The Firmware MAY issue a TPM2_SelfTest command.
5. If the TPM2_SelfTest command immediately returns success and the TPM performs the Self-Test actions associated with the command asynchronously, the FW MAY launch the OS Resume Vector while the TPM2_SelfTest Self-Test actions are still in progress.

# 8 ACPI

**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. Note the TPM device for TPM 2.0 is different than 1.2. According to the ACPI Specification (version 5, Errata A, Section 6.1.5 and 6.1.3) a hardware ID or compatibility ID is either a PNP ID with format "AAA####" or ACPI ID with format "NNNN####". The manufacturer ID returned by a TPM2_GetCapability command can be used to set the "AAA" or "NNNN" portion of the ID. The remaining four hexadecimal digits should be set to a value that allows software to differentiate different device classes built by the same manufacturer.

The example below shows a minimal snippet of typical ASL, with a TPM allocated Interrupt 3.

```
Interrupt 3.

Device (TPM) {

      Name (_HID, "MSFT0101")

      Name (_CRS, ResourceTemplate() {

                  Memory32Fixed (ReadWrite, 0xFED40000, 0x5000,)

                  IRQ(Level, ActiveLow, Shared) {3}

      })

}
```

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. This example TPM is allocated Interrupt 5.

```
Device (TPM) {

      Name (_HID, EISAID("IFX0101"))

      Name (_CID, "MSFT0101")

     Name (_CRS, ResourceTemplate() {

          Memory32Fixed (ReadWrite, 0xFED40000, 0x5000,)

          IRQ(Level, ActiveLow, Shared) {5}

          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 hidden, 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**

## 8.1 ACPI Device Object for TPM

1. A TPM device object MUST contain
    a. a _HID object with a value that is formatted according to the ACPI Specification and contains the manufacturer ID of the TPM vendor and a unique identifier and is in the format “AAA####” or “NNNN####”, and
    b. a _CID object with the value of “MSFT0101” or evaluate to a package where the value “MSFT0101” is one of the IDs within the package, or
    c. A TPM device object MUST contain a _HID object with the value of “MSFT0101”.

### 8.1.1 TPM Visible

While the TPM device is visible:

1. The platform ACPI namespace MUST contain an ACPI device object in an appropriate scope for the TPM according to Section 8.1.
2. When present
    a. The ACPI device object representing the TPM MUST claim all hardware resources consumed by the TPM. **Note:** This includes any legacy IO ports and other hardware resources.
    b. 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.

### 8.1.2 TPM Hidden but Discoverable

1. The platform ACPI namespace MUST contain an ACPI device object in an appropriate scope for the TPM according to Section 8.1.
2. When present
    a. The ACPI device object representing the TPM MUST claim all hardware resources consumed by the TPM. **Note:** This includes any legacy IO ports and other hardware resources.
    b. 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.

### 8.1.3 TPM Hidden and Not Discoverable

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

## 8.2 ACPI Table Usage

**Start of informative comment**

A system’s firmware uses an ACPI table to identify the system’s TCG capabilities to the OS-present environment. The information in this table is not guaranteed to be valid until the Platform Firmware performs the transition from the pre-OS state to OS-Present state.

The presence of the TPM2 ACPI table together with the ACPI device object 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.

If the TPM device implements an interface that corresponds to the TPM 2.0 Command Response Buffer Interface Specification, the ACPI TPM2 table points to the Control Area memory region.

If the ACPI TPM2 table contains the address and size of the Platform Firmware TCG log, firmware "pins" the memory associated with the Platform Firmware TCG log, and reports this memory as "Reserved" memory via the INT 15h/E820 interface. This is done to ensure that the log area contains the TPM2_PCR_Extend operations performed by the most recent system transition from an Off state. If the log were in reclaimable memory, the Platform 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 operation system on its last boot from S5.

**End of informative comment**

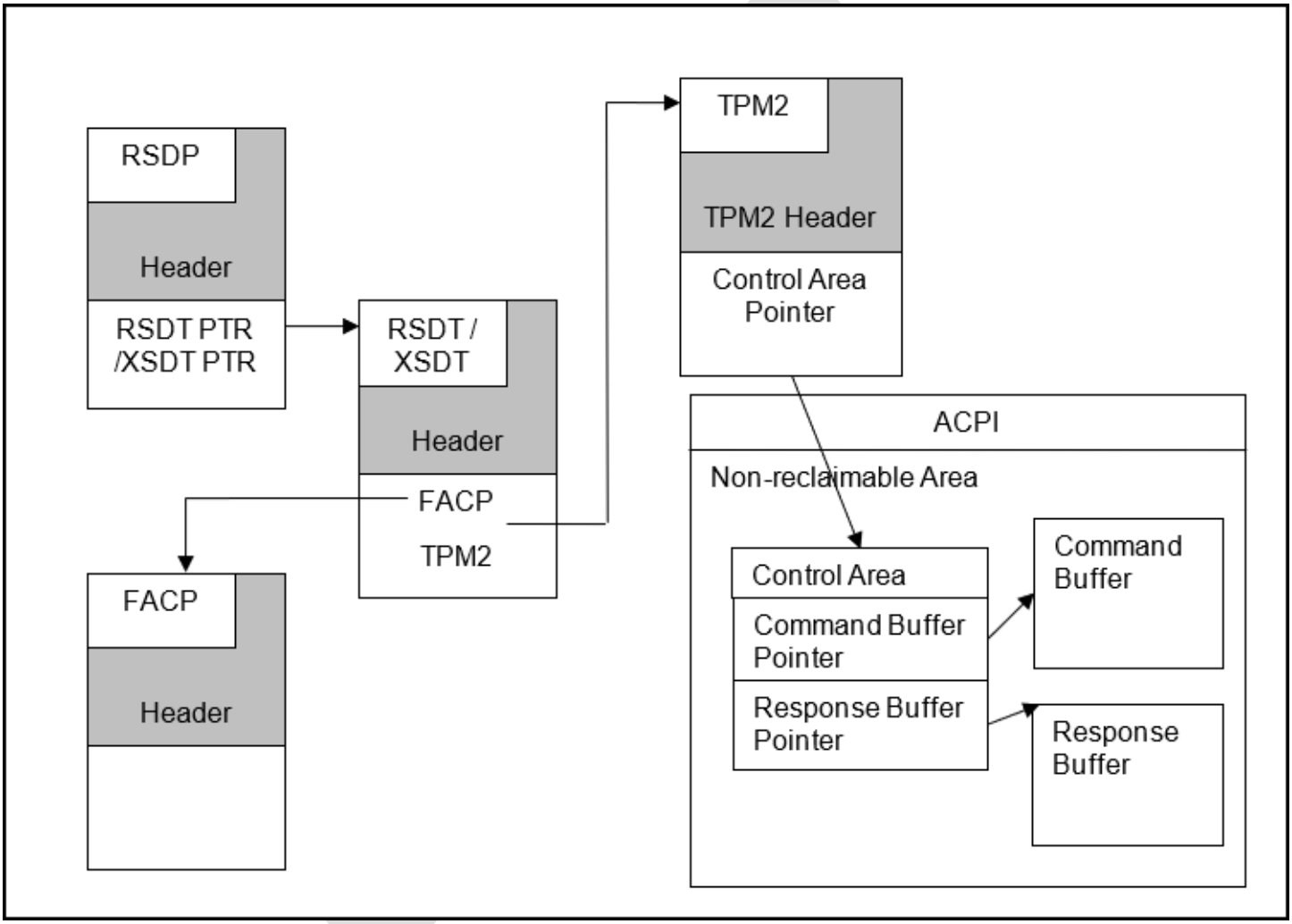


**Figure 10 ACPI Table points to CRB Control Area**

 

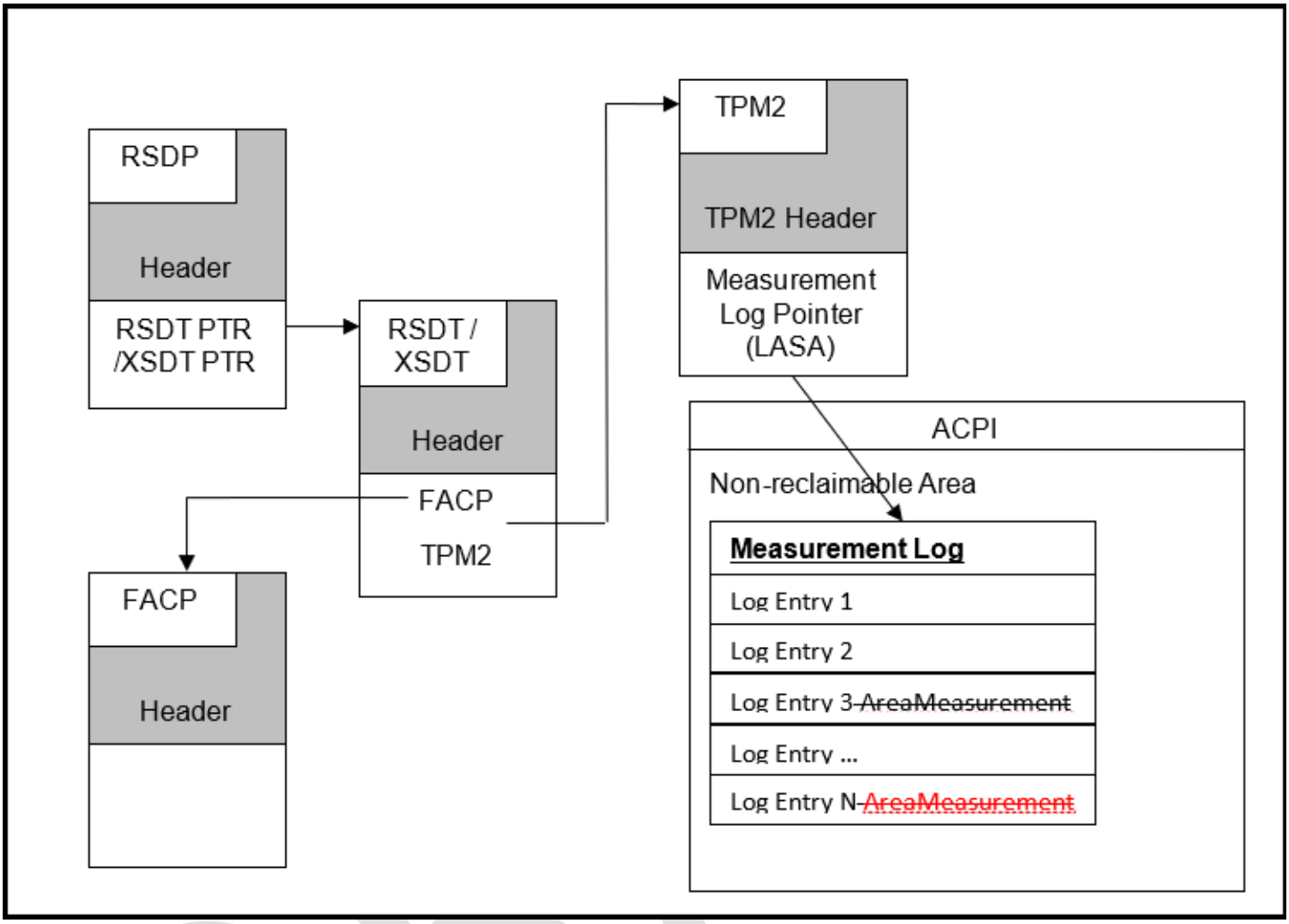


**Figure 11 ACPI Table with pointer to log area**

1. The ACPI table indicated above as “TPM2” is defined in the TCG ACPI Specification version 1.1 revision 37.
2. The ACPI TPM2 table MUST be listed in the RSDT or XSDT ACPI table if the TPM “is discoverable” per the definition above in Section 6 TPM Discoverability, regardless of whether the TPM is currently visible or not. The pointer to the TPM2 table must be correct and the TPM2 table MUST be populated.
3. The ACPI TPM2 table MUST NOT be listed in the RSDT or XSDT ACPI table if the TPM is “not discoverable” per the definition above in Section 6 TPM Discoverability.
4. If the ACPI TPM2 table contains the address and size of the Platform Firmware TCG log:
    a. The Log Area Minimum Length for the TCG event log MUST be at least 64KB.
    **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 Platform Firmware, not a lower bound.
    b. The whole 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.)
    c. The whole memory range of LASA through LASA + (LAML – 1) MUST be initialized to zero by Platform Firmware.

 

d. The OS MAY use the buffer (LASA through LASA + LAML – 1) to store additional event data or other purposes after calling ExitBootServices, even though the TCG EFI Protocol for adding additional log entries is not available.

## 8.3 TPM Interrupt Support

**Start of informative comment**

As usage of the TPM by the OS and by OS-present applications becomes more prevalent, methods of improving TPM performance are increasingly needed. Discrete TPMs have supported interrupts as defined in the PC Client Specific Platform TPM Profile for TPM 2.0, but Platform Firmware has rarely enabled interrupts. Drivers didn't utilize interrupts because the interrupts were not enabled and defined in ACPI by Platform Firmware. This specification requires Platform Firmware to enable interrupts if the TPM supports them. As there is no programmatic method to determine TPM interrupt support, Platform Firmware needs to be preconfigured based on the TPM implemented on the platform. The justification is largely to enhance performance. A TPM driver can, but is not required to, register for an interrupt for command completion rather than polling the TPM interface. The usage is intended for the OS and OS present applications, not for Platform Firmware. An informative example of ASL code, post fix-ups, is contained in the Appendix.

**Note**: FW- and SOC-based TPM's do not have physical interrupt pins, and thus cannot support interrupts.

**End of informative comment**

1. If the TPM supports interrupts, Platform Firmware SHALL declare them in the ACPI table and allow configuration.

# 9 Event Logging

## 9.1 Introduction

**Start of informative comment**

The Event Log is the informational record of measurements made to PCRs by the Platform Firmware, with some events not extended to PCRs. The informational events are used to convey valuable but untrusted information to an evaluator of the log. Each measurement made, as well as the information events, are recorded in the event log as individual entries with specified fields (see Section 9 Event Logging). The first event in the event log is the informational event TCG_EfiSpecIdEvent (See Section 9.4.5.1 Specification ID Version Event). As parsers need a way to determine the format of the events in the log, the first event identifies the version of the log, which implies the format of the events, and the number and size of the recorded digests. The layout of the TCG_PCR_EVENT2 structure that is used for the rest of the event log depends on the information in the first event. For that reason, the first event is formatted as TCG_PCClientPCREvent structure, which is inherited from the Conventional Spec (defined in Section 9.2.1 TCG_PCClientPCREvent Structure) and is independent of digest sizes. Previously, the TCG PC Client Implementation Specification for Conventional BIOS and the TCG PC Client EFI Platform Specification mandated the TCG_EfiSpecIdEvent to be the first event in the event log, so this specification provides the version and digest information in the TCG_EfiSpecIdEvent.

PC Client firmware specifications for TPM 1.2 enabled platforms defined an event log that mandated the use of SHA1 to hash event data and extend the 20 byte digest into PCRs. This specification refers to this log format as the SHA1 log format. With TPM 2.0, PCRs may support other hashing algorithms besides SHA1. If Platform Firmware extends digests to PCRs using other hashing algorithms, an event in the event log has to contain all of the recorded digests. This specification refers to this log format as a crypto agile log format or just as the event log.

A single event in the event log always starts with a header that consists of the PCR index, an event type, the recorded digest(s), and a size of the event data. The PCR index identifies the PCR into which one or more digest of the event data has been extended. The event type identifies the type of event that is recorded in the event log entry. The digest(s) field contains the digest values of the hashed information that is extended into the PCR, note that the digests values may occur in any order within an event. Finally, the event data size defines the size of additional event data. For the SHA1 log format, the digest is a fixed size field of 20 bytes. For the crypto agile log format, the digest consists of a TPML_DIGEST_VALUES (defined in the TPM2 Library Specification Part 2) structure. TCG_EfiSpecIDEvent is formatted as a TCG_PCClientPCREvent structure which is inherited from the Conventional Spec (defined in Section 9.2.1 TCG_PCClientPCREvent Structure). Since this event contains the information needed for parsers to deal with the remaining events in the event log, it is placed first.

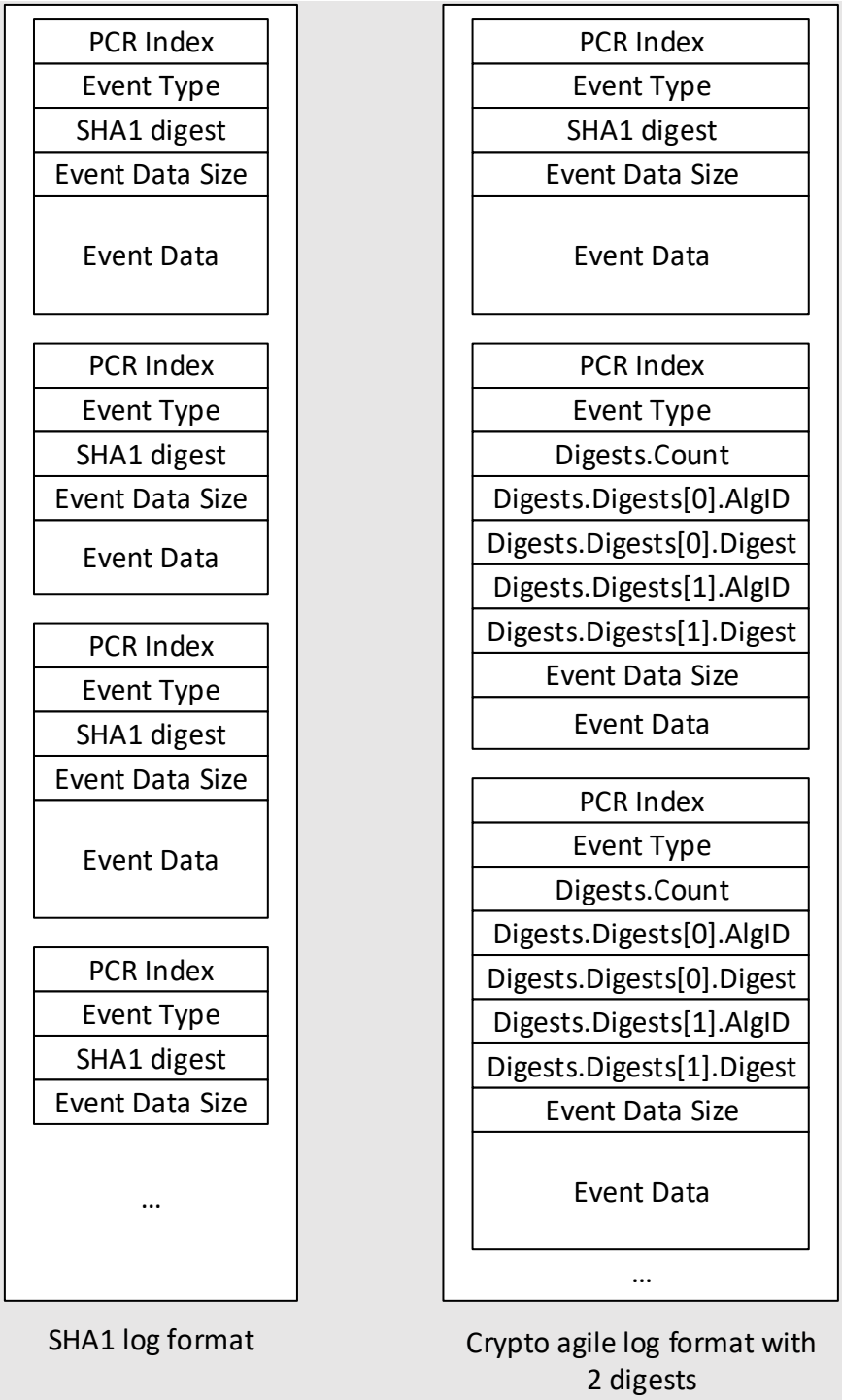


**Figure 12 Depiction of Log Formats**

Note that for the digests in the crypto agile log format, although the type names from the TPM 2.0 Library Specification are used (e.g. TPML_DIGEST_VALUES), the storage of Integers in the event log is little-endian. In this case, the Count and the AlgID fields, leveraged from the TPM 2.0 Library specification, are stored as little-endian. If Platform Firmware utilizes the TPM2_PCR_Event command to hash the data, this command returns a TPML_DIGEST_VALUES structure encoded in big-endian. Platform Firmware would need to modify the

TPML_DIGEST_VALUES structure so that count and AlgID fields are little-endian before adding the event to the event log. Likewise, if Platform Firmware is preparing the TPML_DIGEST_VALUES structure to send to the TPM using a TPM2_PCR_Extend command, the values must be big-endian for the TPM structure. There may be other instances where this type of re-encoding is required. This specification does not produce an exhaustive list.

An event will be densely packed. That is, even though there can be multiple digests in an event, the algorithm ID of the next digest follows immediately after the last byte of the previous digest, without padding. See the example in Table 3 below for an illustration of the offsets in the event log.

To illustrate the crypto agile log event format, here is an EV_SEPARATOR event as example:

**Table 3 Crypto Agile Event Log Event Example 1**

| Field Name | Offset | Size (in bytes) | Content |
|---|---|---|---|
| PCRIndex | 0x00 | 4 | 2 |
| eventType | 0x04 | 4 | 0x00 00 00 04<br>(EV_SEPARATOR) |
| Digests | 0x08 | | |
| Digests.Count | 0x08 | 4 | 1 |
| Digests.Digests | 0x0C | | |
| Digests.Digests[0].AlgID | 0x0C | 2 | 0x00 04 (SHA1) |
| Digests.Digests[0].Digest | 0x0E | 20 | 0x90, 0x69 0xca, 0x78, 0xe7, 0x45, 0x0a, 0x28, 0x51, 0x73, 0x43, 0x1b, 0x3e, 0x52, 0xc5, 0xc2, 0x52, 0x99, 0xe4, 0x73 |
| EventSize | 0x22 | 4 | 4 |
| Event | 0x26 | 4 | 0x00, 0x00, 0x00, 0x00 |

The encoding as a byte stream would look as follows. The start of the line describes the offset for the first byte in the line. (All values in hexadecimal.)

0000: 02 00 00 00 04 00 00 00 – 01 00 00 00 04 00 90 69

0010: ca 78 e7 45 0a 28 51 73 – 43 1b 3e 52 c5 c2 52 99

0020: e4 73 04 00 00 00 00 00 – 00 00

Table 4 is the same separator event using two PCR banks:

**Table 4 Crypto Agile Event Log Event Example**

| Field Name | Offset | Size (in bytes) | Content |
|---|---|---|---|
| PCRIndex | 0x00 | 4 | 2 |
| eventType | 0x04 | 4 | 0x00 00 00 04<br>(EV_SEPARATOR) |
| Digests | 0x08 | | |
| Digests.Count | 0x08 | 4 | 2 |
| Digests.Digests | 0x0C | | |
| Digests.Digests[0].AlgID | 0x0C | 2 | 0x00 04 (SHA1) |
| Digests.Digests[0].Digest | 0x0E | 20 | 0x90, 0x69 0xca, 0x78, 0xe7, 0x45, 0x0a, 0x28, 0x51, 0x73, 0x43, 0x1b, 0x3e, 0x52, 0xc5, 0xc2, 0x52, 0x99, 0xe4, 0x73 |
| Digests.Digests[1].AlgID | 0x22 | 2 | 0x00 0b (SHA-256) |
| Digests.Digests[1].Digest | 0x24 | 32 | 0xdf, 0x3f, 0x61, 0x98, 0x04, 0xa9, 0x2f, 0xdb, 0x40, 0x57, 0x19, 0x2d, 0xc4, 0x3d, 0xd7, 0x48, 0xea, 0x77, 0x8a, 0xdc, 0x52, 0xbc, 0x49, 0x8c, 0xe8, 0x05, 0x24, 0xc0, 0x14, 0xb8, 0x11, 0x19 |
| EventSize | 0x44 | 4 | 4 |
| Event | 0x48 | 4 | 0x00, 0x00, 0x00, 0x00 |

The second example as byte stream looks like this:

0000: 02 00 00 00 04 00 00 00 – 02 00 00 00 04 00 90 69

0010: ca 78 e7 45 0a 28 51 73 – 43 1b 3e 52 c5 c2 52 99

0020: e4 73 0b 00 df 3f 61 98 – 04 a9 2f db 40 57 19 2d

0030: c4 3d d7 48 ea 77 8a dc – 52 bc 49 8c e8 05 24 c0

 

0040: 14 b8 11 19 04 00 00 00 – 00 00 00 00

Note that the algorithm ID of the SHA-256 digest at offset 34 follows directly after the last meaningful byte of the SHA-1 digest. Also, the event data size field (EventSize) at offset 68 follows directly after the last meaningful byte of the SHA-256 digest.

Information events have an event type of value EV_NO_ACTION and PCR index of zero (See Section 9.4.5 EV_NO_ACTION Event Types). The event data field of an EV_NO_ACTION event may contain different data. To distinguish between these different no action events, each EV_NO_ACTION event data starts with a 16 bytes Signature that defines the format of the event data. The TCG_EfiSpecIdEvent event in the crypto agile event log has a SHA1 log format digest field of 20 bytes of zero (See Section 9.4.5.1 Specification ID Version Event). All other EV_NO_ACTION events in a crypto agile log have digest entries for each recorded hashing algorithm and the digests are set to be all zeros.

The 16 bytes Signature for the TCG_EfiSpecIdEvent as defined in this specification is the NUL-terminated ASCII string “Spec ID Event03”. Based on this string an evaluator can determine that this log is formatted as crypto agile log and that the TCG_EfiSpecIdEvent contains information about the recorded digests. For an event log containing SHA1 and SHA256 digests the event may look like this (the vendor specific fields may vary):

**Table 5 TCG_EfiSpecIdEvent Example**

| Field Name | Offset | Size(in bytes) | Content |
|---|---|---|---|
| PCRIndex | 0x00 | 4 | 0x00 00 00 00 |
| eventType | 0x04 | 4 | 0x00 00 00 03<br>(EV_NO_ACTION) |
| Digest | 0x08 | 20 | 20 bytes of zeros |
| EventSize | 0x1C | 4 | 0x00 00 00 25 |
| Event | 0x20 | 37 | TCG_EfiSpecIdEvent |
| TCG_EfiSpecIdEvent<br>.Signature | 0x20 | 16 | “Spec ID Event03” |
| TCG_EfiSpecIdEvent<br>.platformClass | 0x30 | 4 | 0x00 00 00 00 |
| TCG_EfiSpecIdEvent<br>.specVersionMinor | 0x34 | 1 | 0x00 |
| TCG_EfiSpecIdEvent<br>.specVersionMajor | 0x35 | 1 | 0x02 |
| TCG_EfiSpecIdEvent<br>.specErrata | 0x36 | 1 | 0x02 |
| TCG_EfiSpecIdEvent | 0x37 | 1 | 0x02 (UINT64) |

 

| | | | |
|---|---|---|---|
| .uintnSize | | | |
| TCG_EfiSpecIdEvent .NumberOfAlgorithms | 0x38 | 4 | 0x00 00 00 02 (note, this is the Count field) |
| TCG_EfiSpecIdEvent .digestSizes[0].AlgId | 0x3C | 2 | 0x00 04 (SHA1) |
| TCG_EfiSpecIdEvent .digestSizes[0].DigestSize | 0x3E | 2 | 20 |
| TCG_EfiSpecIdEvent .digestSizes[1].AlgId | 0x40 | 2 | 0x00 0b (SHA256) |
| TCG_EfiSpecIdEvent .digestSizes[1].DigestSize | 0x42 | 2 | 32 |
| TCG_EfiSpecIdEvent .vendorInfoSize | 0x44 | 1 | 0x00 |

**End of informative comment**

1. Platform Firmware SHALL create an event log.
    a. This event log SHALL contain all events measured by Platform Firmware.
2. Platform Firmware SHALL create a secondary event log of type EFI_TCG2_FINAL_EVENTS_TABLE, as defined in the TCG EFI Protocol Specification v1.0 revision 9 Section 7 (Log Entries after Get Event Log Service).
    a. This event log SHALL contain all events measured as defined in the TCG EFI Protocol Specification.
3. The first event log entry SHALL be a TCG_PCClientPCREvent structure. See Section 9.2.1 TCG_PCClientPCREvent Structure.
4. The first log entry in the measurement log MUST start at the Measurement Log Start Address. Subsequent measurements MUST be contiguous.
5. Event log entries after the first entry SHALL be TCG_PCR_EVENT2 structures. See Section 9.2.2 TCG_PCR_EVENT2 Structure.
6. For each Hash algorithm enumerated in the TCG_PCClientPCREvent entry, there SHALL be a corresponding digest in all TCG_PCR_EVENT2 structures. **Note:** This includes EV_NO_ACTION events which do not extend the PCR.
7. Sizes for all Hash algorithms included in a TCG_PCR_EVENT2 structure SHALL be enumerated in the TCG_PCClientPCREvent structure.
8. There SHALL be a Hash algorithm in the TCG PCClientPCREvent structure for all allocated PCR banks.
9. There MAY be Hash algorithms in the TCG_PCClientPCREvent structure that do not correspond to allocated PCR banks. See Section 11 Predictive Event Logs.
10. There SHALL NOT be padding between event log entries.
11. All integer structure members SHALL be marshaled in little endian.

## 9.2 TCG Defined Structures

**Start of informative comment**

Some of the data structures used here are derived from UEFI data structures. To simplify parsing of the UEFI data structures, members of type UINTN have been replaces by members of type UINT64.

**End of informative comment**

### 9.2.1 TCG_PCClientPCREvent Structure

**Start of informative comment**

This structure is inherited from the TCG PC Client Implementation Specification for Conventional BIOS. It is only used for the Specification Id Event structure defined in Section 9.4.5.1 Specification ID Version Event.

**End of informative comment**

```
typedef tdTCG_PCClientPCREvent {
      UINT32                pcrIndex;
      UINT32                eventType;
      BYTE                  digest[20];
      UINT32                eventDataSize;
      BYTE                  event[eventDataSize];
} TCG_PCClientPCREvent;
```

**Table 6 TCG_PCClientPCREvent Structure**

| Type | Name | Description |
|---|---|---|
| UINT32 | pcrIndex | The PCR Index to which this event is extended |
| UINT32 | eventType | SHALL be an EV_NO_ACTION event, see Section 9.4.1 Event Types. |
| BYTE[20] | digest | SHALL be 20 Bytes of 0x00 |
| UINT32 | eventDataSize | The size of the event |
| BYTE[eventDataSize] | event | SHALL be a TCG_EfiSpecIdEvent, see Section 9.4.5.1 Specification ID Version Event |

### 9.2.2 TCG_PCR_EVENT2 Structure

**Start of informative comment**

Each “Measurement Log” entry shown in Figure 10 ACPI Table points to CRB Control Area ACPI Table points to CRB Control Area is an instance of this TCG_PCR_EVENT2 structure, except for the Specification ID Version Event, which uses a TCG_PCClientPCREvent{} structure (see Section 9.2.1 TCG_PCClientPCREvent Structure) to allow software which is not TPM 2.0 aware to parse the log. For software parsing the event log, the parser can choose an arbitrary maximum size, but this specification recommends a maximum value for the TCG_PCR_EVENT2.eventSize field of 1MB.

**End of informative comment**

Firmware SHALL construct a list of digests for all enabled PCR banks using the following structure:

 

```
typedef tdTPML_DIGEST_VALUES {
      UINT32      count;
      TPMT_HA     digests[count];
} TPML_DIGEST_VALUES;
```

**Table 7 TPML_DIGEST_VALUES Structure**

| Type | Name | Description |
|---|---|---|
| UINT32 | count | The number of digests in the list |
| TPMT_HA | digests | The list of tagged digests, as sent to the TPM as part of a TPM2_PCR_Extend or as received from a TPM2_PCR_Event command |

Each entry in the Event Log, except the TCG_EfiSpecIdEvent{} structure which is the first event (see Section 9.4.5.1 Specification ID Version Event), SHALL use the following structure:

```
typedef struct tdTCG_PCR_EVENT2{
      UINT32                      pcrIndex;
      UINT32                      eventType;
      TPML_DIGEST_VALUES          digests;
      UINT32                      eventSize;
      BYTE                        event[eventSize];
} TCG_PCR_EVENT2;
```

**Table 8 TCG_PCR_EVENT2 Structure**

| Type | Name | Description |
|---|---|---|
| UINT32 | pcrIndex | The PCR Index to which this event was extended. |
| UINT32 | eventType | Defined in Section 9.4.1 Event Types. |
| TPML_DIGEST_VALUES | digests | A counted list of tagged digests, which contain the digest of the event data (or external data) for all active PCR banks. |
| UINT32 | eventSize | The size of the event data. |
| BYTE | event | The data of the event. |

 

### 9.2.3 UEFI_IMAGE_LOAD_EVENT Structure

**Start of informative comment**

This structure is used in measuring a PE/COFF image. The structure members are defined in the UEFI Specification. The typographic convention for structure member names follows the convention defined in the UEFI specification, not the TCG specification.

**End of informative comment**

```
typedef struct tdUEFI_IMAGE_LOAD_EVENT {
      UEFI_PHYSICAL_ADDRESS  ImageLocationInMemory; // PE/COFF image
      UINT64                 ImageLengthInMemory;
      UINT64                 ImageLinkTimeAddress;
      UINT64                 LengthOfDevicePath;
      UEFI_DEVICE_PATH       DevicePath[LengthOfDevicePath];
                                   // SeeUEFI spec for
                                   // the encodings.
} UEFI_IMAGE_LOAD_EVENT;
```

### 9.2.4 Measuring Industry Standard Tables and Data Structures

**Start of informative comment**

A UEFI platform may support several industry-standard tables and data structures. These include, but are not limited to, ACPI, SMBIOS, and so on. To enable Verifiers to understand what table is measured per event, a new structure has been defined in this revision of the specification. Platform Firmware implementers are recommended to use the new structure. The UEFI_HANDOFF_TABLE_POINTERS structure has been deprecated. Platform Firmware should not mix the structures in the same event log.

From the UEFI Specification, UEFI_CONFIGURATION_TABLE must be:

```
typedef struct {
           EFI_GUID         VendorGuid;
           VOID             *VendorTable;
}UEFI_CONFIGURATION_TABLE;
```

**End of informative comment**

All industry standard tables measured by platform firmware SHALL use UEFI_HANDOFF_TABLE_POINTERS2.

```
typedef struct tdUEFI_HANDOFF_TABLE_POINTERS2 {
UINT8                           TableDescriptionSize;
BYTE[TableDescriptionSize]      TableDescription;
```

 

```
UINT64                          NumberOfTables;
UEFI_CONFIGURATION_TABLE        TableEntry[NumberOfTables];
} UEFI_HANDOFF_TABLE_POINTERS2;
```

Where TableDescription is a Platform-Manufacturer-defined string.

**Start of informative comment:**

This structure has been deprecated.

typedef struct tdUEFI_HANDOFF_TABLE_POINTERS {

UINT64 NumberOfTables;

UEFI_CONFIGURATION_TABLE TableEntry[NumberOfTables];

} UEFI_HANDOFF_TABLE_POINTERS;

**End of informative comment**

### 9.2.5 UEFI_PLATFORM_FIRMWARE_BLOB Structure Definition

**Start of informative comment**

The original UEFI_PLATFORM_FIRMWARE_BLOB structure as defined does not provide information enabling a Verifier to understand the measurement and has been deprecated. This revision of the PFP defines a new structure: UEFI_PLATFORM_FIRMWARE_BLOB2 which adds a Platform-Manufacturer-defined string to describe the measurement. The UEFI_PLATFORM_FIRMWARE_BLOB structure may be deprecated in a future revision.

**End of informative comment**

1. When the CRTM measures the Platform Firmware from system board ROM that loads and launches UEFI Boot Services and UEFI Run Time Services, the CRTM MUST measure the code contained in the Host Platform system board firmware.
2. The CRTM MUST also add an entry of event type EV_S_CRTM_CONTENTS to the Event Log (see Table 7-1) to record the measurement from step 1.
3. Other system board code beyond CRTM content MUST also be measured using event type EV_POST_CODE.
4. Below is the definition of the UEFI_PLATFORM_FIRMWARE_BLOB2 structure that the CRTM MUST put into the Event Log entry TCG_PCR_EVENT2.event[1] field for event types EV_POST_CODE, EV_S_CRTM_CONTENTS, and EV_EFI_PLATFORM_FIRMWARE_BLOB2.

```
typedef struct tdUEFI_PLATFORM_FIRMWARE_BLOB2 {
UINT8                           BlobDescriptionSize;
BYTE[BlobDescriptionSize]       BlobDescription;
UEFI_PHYSICAL_ADDRESS           BlobBase;
UINT64                          BlobLength;
} UEFI_PLATFORM_FIRMWARE_BLOB2;
```

**Start of Informative comment:**

 

This structure has been deprecated.

```
typedef struct tdUEFI_PLATFORM_FIRMWARE_BLOB {
UEFI_PHYSICAL_ADDRESS   BlobBase;
UINT64                  BlobLength;
} UEFI_PLATFORM_FIRMWARE_BLOB;
```

**End of informative comment**

### 9.2.6 Measuring UEFI Variables

**Start of informative comment**

This section defines the event data structures associated with the measurement of UEFI variables.

EFI variables are key/value pairs that consist of identifying information with attributes (the key) and arbitrary data (the value) used to store information passed between the platform and UEFI applications such as OS loaders. See section 7.2 of the UEFI Specification for more information.

The only UEFI variables requiring measurement are those that effect boot policy (that is, where to get the OS Loader), and if UEFI Secure Boot is enabled those that affect the UEFI Secure Boot policy as described in Section 2.3.4.8 PCR[7] – Secure Boot Policy Measurements. Other UEFI variables, such as language, or private variables (that is, GUIDs not defined in the UEFI Specification) are measured into the appropriate PCR, depending upon usage (that is, into PCR[1], PCR[3], or PCR[5]); for more information, see Sections 2.3.4.2 PCR[1] – Host Platform Configuration, 2.3.4.4 PCR[3] – EFI Driver/Application Configuration, and 2.3.4.6 PCR[5] – Boot Manager Configuration and Data of this specification. Even for boot variables that point to the same UEFI application but with different optional data, the measurements are distinct as the measurement will cover the entire UEFI_LOAD_OPTION.

EFI, unlike conventional BIOS, does not need active partition table flags to dictate which OS loader to choose. The OS loader choice is mediated by the UEFI boot options in variables. But the disk partition topology is still important to reflect the system configuration. This configuration information is contained in a UEFI_GPT_DATA structure. The event type EV_EFI_GPT_EVENT designates the measurement of this on-disk geometry, and the event log data structure is described below.

**End of informative comment**

For Event types = EV_EFI_VARIABLE_DRIVER_CONFIG and EV_EFI_VARIABLE_BOOT, the event log entries share a common data structure, namely UEFI_VARIABLE_DATA. EV_EFI_VARIABLE_DRIVER_CONFIG is used to designate the measurement of any UEFI variable, with the exception of the boot variables listed below.

```
typedef struct tdUEFI_VARIABLE_DATA {
  UEFI_GUID   VariableName;
  UINT64      UnicodeNameLength;
  UINT64      VariableDataLength;
  CHAR16      UnicodeName[];
  INT8        VariableData[];   // Driver or platform-specific data
} UEFI_VARIABLE_DATA;
```

 

For the boot variable measurement, the data to be recorded are precisely described in the UEFI specification. Specifically, there is event type EV_EFI_VARIABLE_BOOT. The UEFI firmware MUST measure BootOrder and UEFI Boot#### variables. The event structure for this measurement shares UEFI_VARIABLE_DATA.

```
typedef struct {
  UEFI_PARTITION_TABLE_HEADER UEFIPartitionHeader;
  UINT64                      NumberOfPartitions;
  UEFI_PARTITION_ENTRY        Partitions [NumberOfPartitions];
} UEFI_GPT_DATA;
```

### 9.2.7 DEVICE_SECURITY_EVENT_DATA Structure

**Start of informative comment**

This section defines the event structure for measurements associated with an add-in device, e.g. a PCIE adapter or NVME storage device, that support the "GET_MEASUREMENTS" functionality defined in the DMTF SPDM Specification. This functionality enables a caller to query a device for the measurements of embedded firmware code and configuration.

The DEVICE_SECURITY_EVENT_DATA_HEADER structure contains the measurement and hash algorithm identifier returned by the SPDM "GET_MEASUREMENTS" function without modification. The DeviceType field is populated based on which device layer is used to exchange the SPDM message. If the SPDM message is exchanged via the PCI protocol defined in the PCI specification, the PCI device context is used. If the SPDM message is exchanged via a USB protocol as defined in the USB specifications, the USB device context is used. If a PCI-USB card is present in the system, the card device path is presented as /PciHostBridge/PciBridge/PCIUsbController. Assuming the system uses the PCI protocol to exchange the SPDM messages with this card, the PCI device context should be used.

**End of informative comment**

```
typedef struct {
  UINT8                       Signature[16];
  UINT16                      Version;
  UINT16                      Length;
  UINT32                      SpdmHashAlgo;
  UINT32                      DeviceType;
  SPDM_MEASUREMENT_BLOCK  SpdmMeasurementBlock;
} DEVICE_SECURITY_EVENT_DATA_HEADER;
```

**Table 9 DEVICE_SECURITY_EVENT_DATA_HEADER Definition**

| Type | Name | Description |
|---|---|---|
| UINT8 [16] | Signature | The NUL-terminated ASCII String “SPDM Device Sec” |

 

| | | |
|---|---|---|
| | | SHALL be:<br>{ 0x53 0x50 0x44 0x4d 0x20 0x44 0x65 0x76 0x69 0x63 0x65 0x20 0x53 0x65 0x63 0x00 } |
| UINT16 | Version | Version of this structure, SHALL be “0” |
| UINT16 | Length | Length of the entire DEVICE_SECURITY_EVENT_DATA structure |
| UINT32 | SpdmHashAlgo | The SpdmHashAlgo returned from the device, as defined in the SPDM Specification |
| UINT32 | DeviceType | 0 – No Device Context<br>1 – PCI Device Context<br>2 – USB Device Context<br>3-FF – reserved for future use |
| SPDM_MEASUREMENT_BLOCK | SpdmMeasurementBlock | The SPDM measurement block header returned from the device, as defined by the DMTF SPDM Specification |

```
typedef struct {
  UINT16  Version;
  UINT16  Length;
  UINT16  VendorId;
  UINT16  DeviceId;
  UINT8   RevisionID;
  UINT8   ClassCode[3];
  UINT16  SubsystemVendorID;
  UINT16  SubsystemID;
} DEVICE_SECURITY_EVENT_DATA_PCI_CONTEXT;
```

**Table 10 DEVICE_SECURITY_EVENT_DATA_PCI_CONTEXT** Definition

| Type | Name | Description |
|---|---|---|
| UINT16 | Version | Version of this structure, SHALL be “0” |
| UINT16 | Length | Length of this structure |
| UINT16 | VendorId | PCI VendorId |
| UINT16 | DeviceId | PCI DeviceId |

 

| | | |
|---|---|---|
| UINT8 | RevisionId | PCI RevisionId |
| UINT8[3] | ClassCode | PCI ClassCode |
| UINT16 | SubsystemVendorId | PCI SubsystemVendorId |
| UINT16 | SubsystemId | PCI SubsystemId |

```
Typedef struct {
UINT16        Version;
UINT16        Length;
USB_DEVICE_DESCRIPTOR    DeviceDescriptor;
USB_BOS_DESCRIPTOR       BosDescriptor;
USB_CONFIG_DESCRIPTOR    Configuration1Descriptor;
USB_CONFIG_DESCRIPTOR    Configuration2Descriptor;
…                    ...;
USB_CONFIG_DESCRIPTOR    ConfigurationNDescription;
} DEVICE_SECURITY_EVENT_DATA_USB_CONTEXT;
```

**Table 11 DEVICE_SECURITY_EVENT_DATA_USB_CONTEXT** Description

| Type | Name | Description |
|---|---|---|
| UINT16 | Version | Version of this structure, SHALL be “0” |
| UINT16 | Length | Length of this structure |
| USB_DEVICE_DESCRIPTOR | DeviceDescriptor | USB Device Descriptor, as defined by USB specification.<br>The number of the configuration is included in the Device Descriptor. |
| USB_BOS_DESCRIPTOR | BosDescriptor | Complete BOS Descriptor (if present), as defined by USB specification. |
| USB_CONFIG_DESCRIPTOR | Configuration1Descriptor | Complete Configuration 1 Descriptor, as defined by USB specification. |
| USB_CONFIG_DESCRIPTOR | Configuration2Descriptor | Complete Configuration 2 Descriptor (if present) |
| … | … | … |
| USB_CONFIG_DESCRIPTOR | ConfigurationNDescriptor | Complete Configuration n Descriptor (if present) |

 

```
typedef struct {
  DEVICE_SECURITY_EVENT_DATA_HEADER         EventDataHeader;
  DEVICE_SECURITY_EVENT_DATA_DEVICE_CONTEXT DeviceContext;
} DEVICE_SECURITY_EVENT_DATA;
```

## 9.3 Measurement Event Entries and Log

**Start of informative comment**

The value within a PCR is used both for Sealed Storage and for attestation. When used for attestation, the raw hash value carries little meaning. Therefore, more meaningful structures are used that carry with them information. This information is contained within the structure TCG_PCR_EVENT2.event as described in Section **Error! Reference source not found.**

The procedure for creating the measurement is to perform a hash of the data contained within the TCG_PCR_EVENT2.event field for each supported hash algorithm and construct a TPMT_HA structure. The resulting tagged hashes are placed into the TCG_PCR_EVENT2.digests field and used as the data in the TPM2_PCR_Extend operation. Integer fields in the resulting TPMT_HA structure are encoded in little endian format.

These structures are stored as an unstructured array within the Measured Boot Log. None of the Pre-OS entities, including ACPI, are required to interpret this data. A Boot Manager, a Pre-OS Authentication Application or an OS can obtain a copy of the log using the UEFI Get_Event_Log API defined in the EFI Protocol Specification. Once the OS controls the Host Platform, it 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 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 hidden.

After the invocation of UEFI Get_Event_Log the system may generate additional events, including the results of the EVT_SIGNAL_EXIT_BOOT_SERVICES (EBS). EBS terminates all boot services so that the UEFI Get Event Log function is no longer usable, thus the need to have a declarative variant of the events in a data structure that persists into OS runtime.

**End of informative comment**

1. Platform Firmware SHALL use the hash identified by the algorithmID field recorded by Platform Firmware in the TCG_EfiSpecIdEvent Structure as specified in Section 9.4.5.1 Specification ID Version Event. There MAY be more than one.
2. Procedure for forming a TCG_PCR_EVENT2 structure SHALL be:
    a. Set A to an instantiation of a TCG_PCR_EVENT2 structure.
    b. Set A.pcrIndex to the index of the PCR to be extended.
    c. Set A.eventType to the specified Event Type defined in Section 9.4.1 Event Types.
    d. Fill A.event with either the data to be measured or the description of data to be measured and set A.eventSize to the size of A.event.
    e. Set B to A.digests, a TPML_DIGEST_VALUES structure:
        i. Set B.count to the number of digests in A.digests
        ii. Create C, a TPMT_HA structure, = B.digests[index] where index starts at 0 and continues to count - 1:
            1. Set C.hashAlg to algorithmID of B.digests[index].
            2. Create C.digest according to A.eventType field as defined in Table 12
            3. Repeat for each implemented hash algorithm
        iii. Create B.digests by appending each instance of C. Similar to the TPMT_HA structure used in TPM commands, the fields are densely packet, i.e., if the result of the hashing algorithm is 20 bytes, the C.digest field is only 20 bytes.

**Note** for steps d and e: 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.

    f. Extend A.pcrIndex using A.digests as the value for each implemented PCR bank. Note: All integers should be encoded in Big Endian, before being sent to the TPM.
    g. Append A to the event log. Note: All integers should be encoded in Little Endian before appending to the event log.
    h. The sequence of the Measurement Log entries MUST be in the same sequence that the events were extended into the TPM PCRs.
3. If the TPM is hidden, as defined in Section 6 TPM Discoverability, Platform Firmware MUST not create an event log nor record any log entries.

## 9.4 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 compliant to this specification.

The value within a PCR is used for sealed storage, attestation, and re-construction of the boot flow. The raw hash value in a PCR is sufficient for sealed storage, but not for attestation or replay.

Event Log entries add value to the raw hash values in PCRs for attestation as well as for re-constructing the events that triggered the measurements into the PCRs.

**End of informative comment**

### 9.4.1 Event Types

**Start of informative comment**

One element in an Event Log entry is TCG_PCR_EVENT2.eventType (see section 9.2.2 TCG_PCR_EVENT2 Structure). Table 12, below, is normative and specifies all the event types that can be used in an Event Log entry for the measurement events specified in this document for a UEFI platform.

The value associated with these UEFI specific platform event types must not overlap with the event type values already defined for other TCG platform architecture specifications.

EFI platforms may also use non-EFI specific events from the PC Client Implementation Specification for Conventional BIOS, including but not limited to, EV_POST_CODE, EV_SEPARATOR, EV_S_CRTM_VERSION, EV_S_CRTM_CONTENTS and EV_NO_ACTION. This specification does not speak to measurement of optional tagged events, as defined in the TCG v1.21 PC Client Implementation Specification for Conventional BIOS, section 10.4.2. A composite platform that supports UEFI and conventional BIOS may have such entries in the Event Log.

The value associated with a UEFI specific platform event type MUST be in the range between 0x80000000 and 0x800000FF, inclusive.

**End of informative comment**

The event types in Table 12 are defined for the field TCG_PCR_EVENT2.eventType. Any event type value not defined in this table SHALL not be used by PC Client Platform Firmware and is reserved for application and OS usage.

**Table 12 Events**

| Value | Label | Description |
|---|---|---|
| 0x00000000 | EV_PREBOOT_CERT | Reserved for future use |

 

| Value | Label | Description |
|---|---|---|
| 0x00000001 | EV_POST_CODE<br><br>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.<br><br>The digest field contains the tagged hash of the code or data to be measured (e.g., POST portion of Platform Firmware) for each PCR bank.<br><br>The event field SHOULD NOT contain the actual code or data, but MAY contain informative information about the POST code.<br><br>For POST code, the event data SHOULD be “POST CODE”.<br><br>For embedded SMM code, the event data SHOULD be “SMM CODE”.<br><br>For ACPI flash data, the event data SHOULD be “ACPI DATA”.<br><br>For BIS code, the event data SHOULD be “BIS CODE”.<br><br>For embedded option ROMs, the event data SHOULD be “Embedded UEFI Driver”.<br><br>See Section 2.3.4.1 PCR[0] – SRTM, POST BIOS, and Embedded Drivers. |
| 0x00000002 | EV_UNUSED | This event type is deprecated and MUST NOT be used. |
| 0x00000003 | EV_NO_ACTION<br><br>This event MUST NOT extend any PCR | Used for PCRs[0,6]<br><br>The digests field MUST contain all 0x00’s for each allocated Hash algorithm .<br><br>The event field contains informative data that was not extended into any PCR. |

| Value | Label | Description |
|---|---|---|
| 0x00000004 | EV_SEPARATOR<br><br>This event MUST extend the PCRs 0 through 7 inclusive. | Used for PCRs[0, 1, 2, 3, 4, 5, 6 and 7].<br><br>Per Section 2.3.2 Error Conditions, used to indicate an error occurred recording the CRTM, POST BIOS, or Embedded Option ROMs. For this situation, the digest field MUST contain the tagged digest of the value 00000001h. The event field is arbitrary to allow platform manufacturers to include an indication of what error occurred or other useful troubleshooting information.<br><br>Per Sections 2.3.4 PCR Usage and 7.2 Procedure for Pre-OS to OS-Present Transition, used to delimit actions taken during the pre-OS and OS environments. For this situation, the digests field MUST contain the tagged hash of the event data for each PCR bank. The event data size MUST be 4. The event field MUST contain the hex value 00000000h or FFFFFFFFh. |
| 0x00000005 | EV_ACTION<br><br>This event MUST extend the PCR | Used for PCRs [1, 2, 3, 4, 5, and 6].<br><br>The digests field contains the tagged hash of the event field for each PCR bank.<br><br>A specific action measured as a string defined in Section 9.4.3 EV_ACTION Event Types. |
| 0x00000006 | EV_EVENT_TAG | Used for PCRs defined for OS and application usage.<br><br>Defined for use by Host Platform Operating System or Software.<br><br>May be used by Platform Firmware to measure Legacy Option ROMS using a TCG_PCClientTaggedEvent structure.<br><br>The event field contains the structure defined in Section (0). The digests field MUST contain the tagged hash of the event data for each bank.<br><br>Note: The event data is arbitrary to allow host platform firmware software a standard event type to report events of their choosing. |

| Value | Label | Description |
|---|---|---|
| 0x00000007 | EV_S_CRTM_CONTENTS<br><br>This event MUST extend the PCR | Used for PCR[0] only.<br><br>The digests field contains the tagged hash of the SRTM for each PCR bank.<br><br>The event field SHOULD NOT contain the actual SRTM code but MAY contain informative information about the SRTM code.<br><br>See Section 2.3.4.1 PCR[0] – SRTM, POST BIOS, and Embedded Drivers. |
| 0x00000008 | EV_S_CRTM_VERSION<br><br>This event MUST extend the PCR | Used for PCR[0] only.<br><br>The digests field contains the tagged hash of the event field for each PCR bank.<br><br>The event field contains the version string of the SRTM.<br><br>See Section 2.3.4.1 PCR[0] – SRTM, POST BIOS, and Embedded Drivers. |
| 0x00000009 | EV_CPU_MICROCODE<br><br>This event MUST extend the PCR | Used for PCR[1] only.<br><br>The digests field contains the tagged hash of the microcode patch applied for each PCR bank.<br><br>The event field contains a descriptor of the microcode patch.<br><br>See Section 2.3.4.2 PCR[1] – Host Platform Configuration. |
| 0x0000000A | EV_PLATFORM_CONFIG_FLAGS<br><br>This event MUST extend the PCR | Used in PCR[1] only.<br><br>The digests field contains the tagged hash of the event field for each PCR bank.<br><br>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.<br><br>See Section 2.3.4.2 PCR[1] – Host Platform Configuration. |

| Value | Label | Description |
|---|---|---|
| 0x0000000B | EV_TABLE_OF_DEVICES<br><br>This event MUST extend the PCR | Used in PCR[1] only.<br><br>The digests field contains the tagged hash of the event field for each PCR bank.<br><br>The event field contents are manufacturer implementation-specific.<br><br>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.<br><br>See Section 2.3.4.2 PCR[1] – Host Platform Configuration. |
| 0x0000000C | EV_COMPACT_HASH<br><br>This event MUST extend the PCR | May be used for any PCRs except 0, 1, 2, or 3.<br><br>The event field MAY be informative or MAY be hashed to generate the digests field, depending on the component recording the event.<br><br>This event is measured using the TCG_CompactHashLogExtendEvent. While it can be used by any component, it is typically used by Boot Manager Code to measure events. The contents of the event field are specified by the caller.<br><br>See Section 2.3.4.5 PCR[4] – Boot Manager Code and Boot Attempts<br><br>This event may also be used by Platform Firmware vendors to measure events into PCR[6].<br><br>See Section 2.3.4.7 PCR[6] – Host Platform Manufacturer Specific Measurements |
| 0x0000000D | EV_IPL | This event is deprecated. |
| 0x0000000E | EV_IPL_PARTITION_DATA | This event is deprecated |

| Value | Label | Description |
|---|---|---|
| 0x0000000F | EV_NONHOST_CODE<br><br>This event MUST extend the PCR | Used for PCR[0,2] only.<br><br>The executable component of any Non-Host Platform.<br><br>The digests field contains the tagged hash of the Non-Host Platform code for each PCR bank.<br><br>The event field may be determined by the platform manufacturer.<br><br>See Sections 2.3.4.1 PCR[0] – SRTM, POST BIOS, and Embedded Drivers and 2.3.4.3 PCR[2]-UEFI Drivers and UEFI Applications. |
| 0x00000010 | EV_NONHOST_CONFIG<br><br>This event MUST extend the PCR | Used for PCR[1,3] only.<br><br>Configuration information or data associated with a Non-Host Platform.<br><br>The digests field contains the tagged hash of the Non-Host Platform configuration information for each PCR bank.<br><br>The event field is defined by the platform manufacturer.<br><br>See Sections 2.3.4.2 PCR[1] – Host Platform Configuration and 2.3.4.4 PCR[3] – EFI Driver/Application Configuration. |
| 0x00000011 | EV_NONHOST_INFO<br><br>This event MUST extend the PCR | Used for PCR[0] only.<br><br>Information about the presence of a Non-Host Platform.<br><br>The digests field contains the tagged hash of the event field for each PCR bank.<br><br>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.<br><br>See Section 2.3.4.1 PCR[0] – SRTM, POST BIOS, and Embedded Drivers. |

| Value | Label | Description |
|---|---|---|
| 0x00000012 | EV_OMIT_BOOT_DEVICE_EVENTS | Used for PCR[4] only.<br>The digests field contains the tagged hash of the event field for each PCR bank.<br>The event field MUST be “BOOT ATTEMPTS OMITTED” in all caps.<br>See Section 2.3.4.5 PCR[4] – Boot Manager Code and Boot Attempts0. |
| 0x00000013-<br>0x0000FFFF | Reserved | Reserved for future use |
| 0x80000000 | EV_EFI_EVENT_BASE | Base value for all UEFI platform event type values below |
| 0x80000001 | EV_EFI_VARIABLE_DRIVER_CONFIG | May be used for PCR[1, 3, 5 or 7]<br>This event is used to measure configuration for EFI Variables.<br>The digests field contains the tagged hash of the variable data, e.g. variable data, GUID, or unicode string for each PCR bank.<br>The event field MUST contain a UEFI_VARIABLE_DATA structure.<br>See Section 9.2.6 Measuring UEFI Variables |
| 0x80000002 | EV_EFI_VARIABLE_BOOT | Used for PCR[ 1] only<br>This event is used to measure boot variables.<br>The digests field contains the tagged Hash of the UEFI Boot#### variables and the BootOrder variable for each PCR bank.<br>The event field MUST contain a UEFI_VARIABLE_DATA structure<br>See Section 9.2.6 Measuring UEFI Variables |

| Value | Label | Description |
|---|---|---|
| 0x80000003 | EV_EFI_BOOT_SERVICES_APPLICATION | Used for PCR[2,4] only<br>This event measures information about the specific application loaded from the Boot Device.<br>The digests field contains the tagged Hash of the normalized code from the loaded UEFI Boot Services application for each PCR bank.<br>The event field MUST contain a UEFI_IMAGE_LOAD_EVENT structure.<br>See Section 9.2.3 UEFI_IMAGE_LOAD_EVENT Structure |
| 0x80000004 | EV_EFI_BOOT_SERVICES_DRIVER | Used for PCR[0,2] only<br>The digests field contains the tagged hash of the normalized code from the loaded UEFI Boot Services driver for each PCR bank.<br>The event field MUST contain a UEFI_IMAGE_LOAD_EVENT structure<br>See Section 9.2.3 UEFI_IMAGE_LOAD_EVENT Structure |
| 0x80000005 | EV_EFI_RUNTIME_SERVICES_DRIVER | Used for PCR[0,2] only<br>This event measures information about embedded and add-in adapters with UEFI drivers.<br>The digests field contains the tagged hash of the normalized code from the loaded UEFI Runtime Services driver for each PCR bank.<br>The event field MUST contain a UEFI_IMAGE_LOAD_EVENT structure.<br>See Section 9.2.3 UEFI_IMAGE_LOAD_EVENT Structure |
| 0x80000006 | EV_EFI_GPT_EVENT | Used for PCR[5] only<br>This event measures the UEFI GPT Table.<br>The digests field contains the tagged hash of the GPT Table for each PCR bank.<br>The event data MUST contain a UEFI_GPT_DATA structure.<br>See Section 9.2.6 Measuring UEFI Variables. |

| Value | Label | Description |
|---|---|---|
| 0x80000007 | EV_EFI_ACTION | Used for PCR[1,2,3,4,5,6,7]<br><br>The digests field contains the tagged hash of Event field for each PCR bank.<br><br>A specific action measured as an EFI string defined in Section 9.4.4 EV_EFI_ACTION Event Types. |
| 0x80000008 | EV_EFI_PLATFORM_FIRMWARE_BLOB | This event has been deprecated.<br><br>See Section 9.2.5 UEFI_PLATFORM_FIRMWARE_BLOB Structure Definition |
| 0x80000009 | EV_EFI_HANDOFF_TABLES | This event has been deprecated.<br><br>See Section 9.2.4 Measuring Industry Standard Tables and Data Structures. |
| 0x8000000A | EV_EFI_PLATFORM_FIRMWARE_BLOB2 | Used for PCR[0, 2, 4] only<br><br>This event measures information about non-PE/COFF images. This allows for non-PE/COFF images in PCR[2] or PCR[4].<br><br>The digests field contains the tagged hash of all the code (PE/COFF .text sections or other) for each PCR bank.<br><br>The event MUST contain a UEFI_PLATFORM_FIRMWARE_BLOB2 structure.<br><br>See Section 9.2.5 UEFI_PLATFORM_FIRMWARE_BLOB Structure Definition |
| 0x8000000B | EV_EFI_HANDOFF_TABLES2 | Used for PCR[1] only<br><br>The digests field contains the tagged hash of the system configuration tables which are referenced by entries in UEFI_HANDOFF_TABLE_POINTERS2 for each PCR bank.<br><br>The event field MUST contain the UEFI table UEFI_HANDOFF_TABLE_POINTERS2.<br><br>See Section 9.2.4 Measuring Industry Standard Tables and Data Structures. |

| Value | Label | Description |
|---|---|---|
| 0x8000000C-0x8000000F | Reserved | Reserved for future use |
| 0x80000010 | EV_EFI_HCRTM_EVENT | Used for PCR[0] only<br>This event is used to record an event for the digest extended to PCR[0] as part of an H-CRTM event.<br>The digests field contains the tagged hash of the H-CRTM event data for each PCR bank<br>The Event Data MUST be the string: “HCRTM”.<br>See Section 2.3.5.1 H-CRTM Localities and Concepts |
| 0x80000011-0x800000DF | Reserved | Reserved for future use. |
| 0x800000E0 | EV_EFI_VARIABLE_AUTHORITY | Used for PCR[7] only<br>This event describes the Secure Boot variables.<br>The digests field contains the tagged hash of the UEFI_SIGNATURE_DATA value from the UEFI_SIGNATURE_LIST used to validate the loaded image for each PCR bank.<br>The event field MUST contain a UEFI_VARIABLE_DATA structure.<br>See Section 9.2.6 Measuring UEFI Variables. |
| 0x800000E1 | EV_EFI_SPDM_FIRMWARE_BLOB | Used for PCR[2] only.<br>This event is used to record an extended digest for the embedded firmware of an add-in device that supports SPDM “GET_MEASUREMENTS” functionality. This event records extended digests of SPDM MeasurementType 0 or 1 only.<br>The digests field contains the tagged digest for each PCR bank. The event field MUST be a DEVICE_SECURITY_EVENT_DATA structure. See Section 9.2.7 DEVICE_SECURITY_EVENT_DATA Structure. |

| Value | Label | Description |
|---|---|---|
| 0x800000E2 | EV_EFI_SPDM_FIRMWARE_CONFIG | Used for PCR[3] only.<br><br>This event is used to record an extended digest of the configuration of embedded firmware for an add-in device that supports SPDM "GET_MEASUREMENTS" functionality. This event records extended digests of SPDM MeasurementType 2 or 3 only.<br><br>The digests field contains the tagged digest for each PCR bank. The event field MUST be a DEVICE_SECURITY_EVENT_DATA structure. See Section 9.2.7 DEVICE_SECURITY_EVENT_DATA Structure. |
| 0x800000E1-0x8000FFFF | Reserved | Reserved for future use |

### 9.4.2 Tagged Event Log Structure

**Start of informative comment**

This event type was originally specified in the TCG PC Client Implementation Specification for Conventional BIOS. It was deprecated in the original version of the PC Client Specific Platform Firmware Profile for TPM 2.0 systems. As a result of more interest in platform attestation services, which are not just confined to the results of the Platform Firmware measurements, but also the host platform OS and software environment, this event type has been redefined in this specification for use by OS and software on a PC Client platform. Other than the Event Type and structure, it is not standardized.

**End of informative comment**

1. Tagged Event Data MUST be measured and logged using the TCG_PCR_EVENT2 structure.
2. The TCG_PCR_EVENT2.eventType SHALL be EV_EVENT_TAG.
3. The TCG_PCR_EVENT2.event SHALL contain one or more of the TCG_PCClientTaggedEvent structures:
4. TCG_PCR_EVENT2.eventSize contains the size of all the TCG_PCClientTaggedEvent structures stored in the TCG_PCR_EVENT2.event field.

```
typedef struct tdTCG_PCClientTaggedEvent{
      UINT32      taggedEventID;
      UINT32      taggedEventDataSize;
      BYTE
      taggedEventData[taggedEventDataSize];
} TCG_PCClientTaggedEvent;
```

 

**Table 13 TCG PC Client Tagged Event Structure**

| Type | Name | Description |
|---|---|---|
| UINT32 | taggedEventID | This is a unique identifier defined by the measuring OS or application. Note: This is Host OS or software defined data and not defined by this specification. |
| UINT32 | taggedEventDataSize | Size of taggedEventData |
| BYTE | taggedEventData | The data of the event. Note: This is Host OS or software defined data and not defined by this specification. |

### 9.4.3 EV_ACTION Event Types

For each EV_ACTION event produced by the platform, Platform Firmware 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, nor a NUL terminator. They SHALL be logged using the following:

TCG_PCR_EVENT2.eventType = 05h (the value for the EV_ACTION event type fromTable 12 in Section 9.4.1 Event Types)

TCG_PCR_EVENT2.digests = the tagged hash (for each PCR bank) of the Event[] field

TCG_PCR_EVENT2.event[] = ASCII string from the following table:

**Table 14 EV_ACTION Event types**

| Action Index[1] | String | Purpose and Comments | PCR |
|---|---|---|---|
| 0 | “Calling Ready to Boot” | BIOS is calling Ready to Boot. If no additional strings are posted in log, that means the Boot Manager did not return control to Platform Firmware. | 4 |

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

| Action Index[1] | String | Purpose and Comments | PCR |
|---|---|---|---|
| 1 | “Returned Ready to Boot” | Entering the Ready to Boot handler. This means either Platform Firmware has transferred control to the Ready to Boot handler; or Platform Firmware received control back from a failed IPL.<br><br>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” | Deprecated | n/a |
| 3 | “Booting BCV Device s” | BIOS is IPL/Booting a BCV Device.<br><br>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.<br><br>May be deprecated in a future version of this specification | 4 |
| 4 | “Booting BEV Device s” | BIOS is IPL/Booting a BEV Device.<br><br>The value ‘s’ is an ASCII string supplied by the BEV device.<br><br>May be deprecated in a future version of this specification | 4 |

| Action Index[1] | String | Purpose and Comments | PCR |
|---|---|---|---|
| 5 | “Entering ROM Based Setup” | BIOS is entering ROM-Based Setup or Option ROM-Based Setup during pre-OS 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 2.3.4.2 PCR[1] – Host Platform Configuration and 2.3.4.4 PCR[3] – EFI Driver/Application Configuration. | 1 or 3 |
| 6 | “Booting to Parties n” | BIOS is IPL/Booting from a PARTIES Partition #n.<br>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.<br>May be deprecated in a future version of this specification. | 4 |
| 7 | “User Password Entered” | Deprecated | 1 |
| 8 | “Administrator Password Entered” | Deprecated | 1 |
| 9 | “Password Failure” | Deprecated | 1 |
| 10 | “Wake Event n” | Deprecated | n/a |
| 11 | “Boot Sequence User Intervention” | User altered the boot sequence. | 1 |
| 12 | “Chassis Intrusion” | The case was opened. | 1 |
| 13 | “Non Fatal Error” | Deprecated | n/a |
| 14 | “Start Option ROM Scan” | Deprecated | n/a |
| 15 | “Unhiding Option ROM Code” | Deprecated | n/a |
| 16 | “<OpRom Specific non-IPL String>” | Deprecated | n/a |

| Action Index[1] | String | Purpose and Comments | PCR |
|---|---|---|---|
| 17 | "<OpRom Specific IPL String>" | Deprecated | n/a |
| 18 | "HDD Password Entered" | Deprecated | 1 |

### 9.4.4 EV_EFI_ACTION Event Types

**Start of informative comment**

The EV_EFI_ACTION event type defined in Table 15, below extends into a specific PCR the measurement of a specific string value that indicates a specific event occurred during the platform or OS boot process.

**Note**: The opening and closing quote characters shown in the String Value column of Table 15 must not be included in the TCG_PCR_EVENT2.event field and must not be included in the input to the measurement function.

**End of informative comment**

Table 15, below, specifies all the specific string values that can be used for EV_EFI_ACTION on a UEFI platform. The strings below are enclosed in quotes for clarity; the actual log entries SHALL NOT include the quote characters or a NUL terminator.

**Table 15 EV_EFI_ACTION Strings**

| Action Index[2] | String | Purpose and Comments | PCR |
|---|---|---|---|
| 1 | "Calling EFI Application from Boot Option" | Boot Manager attempting to execute code from a Boot Option<br>See Section 7.2.4 Measuring OS Boot Events | 4 |
| 2 | "Returning from EFI Application from Boot Option" | Attempt to execute code from Boot Option was unsuccessful<br>See Section 7.2.4 Measuring OS Boot Events | 4 |
| 3 | "Exit Boot Services Invocation" | Boot Manager has sent the call to UEFI to end Boot Services.<br>See Section 7.2.4 Measuring OS Boot Events | 5 |
| 4 | "Exit Boot Services Returned with Failure" | UEFI encountered an error closing Boot Services<br>See Section 7.2.4 Measuring OS Boot Events | 5 |

[2] This is only used for reference in this document.

 

| Action Index[2] | String | Purpose and Comments | PCR |
|---|---|---|---|
| 5 | "Exit Boot Services Returned with Success" | UEFI successfully existed Boot Services and pre-OS environment has been terminated<br><br>See Section 7.2.4 Measuring OS Boot Events | 5 |
| 6 | "UEFI Debug Mode" | If a debugger is launched, this string is used to record the entry into debug mode.<br><br>See Section 7.2.4 Measuring OS Boot Events | 7 |

### 9.4.5 EV_NO_ACTION Event Types

**Start of informative comment**

An EV_NO_ACTION event will not result in a digest being extended into a PCR, but has value to an evaluator of the log. For EV_NO_ACTION events other than the EFI Specification ID event (Section 9.4.5.1 Specification ID Version Event) the log will contain EV_NO_ACTION events in the TCG_PCR_EVENT2 format, so a parser does not have to switch between event formats. To eliminate further conditional treatment of the TCG_PCR_EVENT2 structure in the parser, the TCG_PCR_EVENT2 event for EV_NO_ACTION events will contain a digest entry for each algorithm that is used in other TCG_PCR_EVENT2 events, but the digest values will be all zeros. For example, if only the SHA256 PCR bank is used, the TCG_PCR_EVENT2.digests.count value is 1, TCG_PCR_EVENT2.digests.digests[0].algID is 0xB , and TCG_PCR_EVENT2.digests.digests[0].digest is 32 bytes of zeros. In the normative text the short notation of TCG_PCR_EVENT2.digests = 0 is used to express this.

**End of informative comment**

1. All EV_NO_ACTION events SHALL set TCG_PCR_EVENT2.pcrIndex = 0, unless otherwise specified.
2. All EV_NO_ACTION events SHALL set TCG_PCR_EVENT2.eventType = 03h (the value for the EV_NO_ACTION event type from Table 12 in Section 9.4.1 Event Types).
3. All EV_NO_ACTION events SHALL set TCG_PCR_EVENT2.digests to all 0x00's for each allocated Hash algorithm. See Section 9.2.2 TCG_PCR_EVENT2 Structure.
4. All EV_NO_ACTION events SHALL set TCG_PCR_EVENT2.event to one of the events described in the following sections.
5. For each EV_NO_ACTION event produced by the platform, Platform Firmware MUST create and record the event indicated in this section.
6. EV_NO_ACTION events MUST NOT extend the PCR, but they are recorded in the event log.

#### 9.4.5.1 Specification ID Version Event

**Start of informative comment**

The first event in the event log is the Specification ID version. This event is not extended to a PCR. This event provides information to a consumer of the event log about what version of the specification is implemented by firmware. Because the TCG_PCR_EVENT2 structure contains digest size information which cannot be interpreted by the consumer, the TCG_EfiSpecIdEvent{} structure will be the event field of a TCG_PCClientPCREvent{} (see Section 9.2.1 TCG_PCClientPCREvent Structure) structure with a 20 byte digest field of all zeros. It is an EV_NO_ACTION event type. The SpecVersionMajor, SpecVersionMinor, and SpecErrata fields should be modified to match the version of the PFP with which the Platform Firmware complies.

The TCG_EfiSpecIDEvent{} structure contains one or more TCG_Efi_SpecIdEventAlgorithmSize structures. These contain the algorithmID and digestSize for the measurement algorithms used by firmware. As consumers of the event log may not recognize the algorithmID, the digestSize field allows a parser to consume the events in the log without knowing the implicit size of the algorithm defined by the algorithmID. The information contained in this structure will vary based on the capabilities of the TPM and the platform. If platform firmware does not support the same number of hash algorithms as the TPM has enabled, i.e. TPM enables SHA-1 and SHA-256 but platform firmware only supports SHA-256, platform firmware has two options. For the first option, platform firmware would inform the user and disable the unsupported bank. For the second option, the platform would use the TPM2_Event command (or TPM2_EventSequence, TPM2_SequenceData, and TPM2_SequenceEnd) to send data to the TPM for the TPM to hash and extend to all enabled banks. Because the second option add significant time to the platform boot process, it is not anticipated to be used, but is offered as there are circumstances where it might be needed. The following table provides an example of what might be observed in the structure based on platform support.

**Table 16 EfiSpecIdEventAlgorithmSize Examples**

| TPM Algorithm Support | Platform Algorithm Support | EfiSpecIdEventAlgorithmSize |
|---|---|---|
| Single PCR Bank, Single Algorithm | Single Algorithm | 1 EfiSpecIdEventAlgorithmSize structure |
| Single PCR Bank, Multiple Algorithms | Single Algorithm | 1 EfiSpecIdEventAlgorithmSize structure |
| Single PCR Bank, Multiple Algorithms | Multiple Algorithms | 1 EfiSpecIdEventAlgorithmSize structure |
| 2 PCR Banks, 2 Algorithms | Single Algorithm | 1 EfiSpecIdEventAlgorithmSize structure |
| 2 PCR Banks, 2 Algorithms | Multiple Algorithms | 2 EfiSpecIdEventAlgorithmSize structures |

**End of informative comment**

```
typedef struct tdTCG_EfiSpecIdEventAlgorithmSize {
   UINT16      algorithmId;
   UINT16      digestSize;
} TCG_EfiSpecIdEventAlgorithmSize;
```

**Table 17 TCG_EfiSpecIdEventAlgorithmSize**

| Type | Name | Description |
|---|---|---|
| UINT16 | algorithmID | This value of this field SHALL be the algorithm ID (hashAlg) of the Hash used by BIOS<br><br>Note: Systems that support multiple active PCR banks may report more than one algorithm ID. This field contains the TPMI_ALG_HASH identifier for the implemented Hash algorithm. |

| Type | Name | Description |
|---|---|---|
| UINT16 | digestSize | The size of the digest produced by the implemented Hash algorithm. |

```
typedef struct tdTCG_EfiSpecIdEvent {
   BYTE                          Signature[16];
   UINT32                        platformClass;
   UINT8                         specVersionMinor;
   UINT8                         specVersionMajor;
   UINT8                         specErrata;
   UINT8                         uintnSize;
   UINT32                        numberOfAlgorithms;
   TCG_EfiSpecIdEventAlgorithmSize
                                 digestSizes[numberOfAlgorithms];
   UINT8                         vendorInfoSize;
   BYTE                          vendorInfo[VendorInfoSize];
} TCG_EfiSpecIDEvent;
```

**Table 18 TCG_EfiSpecIdEvent**

| Type | Name | Description |
|---|---|---|
| BYTE[16] | Signature | The NUL-terminated ASCII string ”Spec ID Event03”.<br><br>SHALL be set to {0x53, 0x70, 0x65, 0x63, 0x20, 0x49, 0x44, 0x20, 0x45, 0x76, 0x65, 0x6e, 0x74, 0x30, 0x33, 0x00}. |
| UINT32 | platformClass | The value for the Platform Class. The enumeration is defined in the TCG ACPI Specification Client Common Header. |
| UINT8 | FamilyVersionMinor | The PC Client Platform Firmware Profile Specification Family minor version number this BIOS supports. Any BIOS supporting this version (2.0) MUST set this value to 0x00. |
| UINT8 | FamilyVersionMajor | The PC Client Platform Firmware Profile Specification Family major version number this BIOS supports. Any BIOS supporting this version (2.0) MUST set this value to 0x02. |

 

| | | |
|---|---|---|
| UINT8 | specRevision | The PC Client Platform Firmware Profile Specification Revision number this BIOS supports. Any BIOS supporting this version (1.05) MUST set this value to 0x69 (105 in Hex). |
| UINT8 | uintnSize | Specifies the size of the UINTN fields used in various data structures used in this specification. 0x01 indicates UINT32 and 0x02 indicates UINT64. |
| UINT32 | numberOfAlgorithms | The number of Hash algorithms in the digestSizes field. This field MUST be set to a value of 0x01 or greater. |
| TCG_EfiSpecIdEventAlgorithmSize[numberOfAlgorithms] | digestSizes | Each TCG_EfiSpecIdEventAlgorithmSize SHALL contain an algorithmId and digestSize for each hash algorithm used in the TCG_PCR_EVENT2 structure, the first of which is a Hash algorithmID and the second is the size of the respective digest. |
| UINT8 | vendorInfoSize | Size in bytes of the VendorInfo field. Maximum value MUST be FFh bytes. |
| BYTE[VendorInfoSize] | vendorInfo | Provided for use by Platform Firmware 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. |

#### 9.4.5.2 Firmware Integrity Measurement Reference Manifest Event

**Start of informative comment**

The TCG_Sp800_155_PlatformId_Event structure is deprecated with this revision of the specification. There is no known use of this structure, but to ensure parsers can correctly handle the modifications made in this revision of the PFP, a new structure has been created.

**End of informative comment**

```
typedef struct tdTCG_Sp800_155_PlatformId_Event2 {
  BYTE            Signature[16];
  UINT32          VendorId;
  UEFI_GUID       ReferenceManifestGuid;
  UINT8           PlatformManufacturerStrSize;
```

 

```
    BYTE            PlatformManufacturerStr [PlatformManufacturerStrSize];
    UINT8           PlatformModelSize;
    BYTE            PlatformModel[PlatformModelSize];
    UINT8           PlatformVersionSize;
    BYTE            PlatformVersion[PlatformVersionSize];
    UINT8           PlatformModelSize;
    BYTE            PlatformModel[PlatformModelSize];
    UINT8           FirmwareManufacturerStrSize;
    BYTE            FirmwareManufacturerStr[FirmwareManufacturerStrSize];
    UINT32          FirmwareManufacturerId;
    UINT8           FirmwareVersion;
    BYTE            FirmwareVersion[FirmwareVersionSize]];
} TCG_Sp800-155-PlatformId_Event2;
```

**Table 19 BIM Reference Manifest Event**

| Type | Name | Description |
|---|---|---|
| BYTE[16] | Signature | The NUL-terminated ASCII String “SP800-155 Event”. SHALL be set to {0x53 0x50 0x38 0x30 0x30 0x2d 0x31 0x35 0x35 0x20 0x45 0x76 0x65 0x6e 0x74 0x32} |
| UINT32 | VendorId | Where Vendor ID is an integer defined at http://www.iana.org/assignments/enterprise-numbers |
| EFI_GUID | ReferenceManifestGuid | ReferenceManifestGuid is the 16-byte identifier of a given platform’s static configuration of code. |
| UINT8 | VendorSize | Size in bytes of the Vendor field. Maximum value MUST be FFh bytes. |
| BYTE[VendorSize] | Vendor | The Vendor name as a NUL-terminated ASCII string. |
| UINT8 | ModelSize | Size in bytes of the Model field. Maximum value MUST be FFh bytes. |
| BYTE[ModelSize] | Model | The platform Model Name or Number as a NUL-terminated ASCII string. |

#### 9.4.5.3 Startup Locality Event

**Start of informative comment**

The Startup Locality event should be placed in the log before any event which extends PCR[0]. This allows software which needs to parse the TCG Event Log to initialize its internal PCR[0] state correctly. See Section 2.3.4.1 PCR[0] – SRTM, POST BIOS, and Embedded Drivers.

**End of informative comment**

```
typedef struct tdTCG_EfiStartupLocalityEvent {
   BYTE        Signature[16];
   UINT8       StartupLocality;
} TCG_EfiStartupLocalityEvent;
```

**Table 20 Startup Locality Event**

| Type | Name | Description |
|---|---|---|
| BYTE[16] | Signature | The NUL-terminated ASCII string “StartupLocality” SHALL be set to {0x53 0x74 0x61 0x72 0x74 0x75 |

 

| | | |
|---|---|---|
| | | 0x70 0x4C 0x6F 0x63 0x61 0x6C 0x69 0x74 0x79 0x00} |
| UINT8 | StartupLocality | SHALL be the value of the Startup Locality which sent the TPM2_Startup command.<br>0: Startup Locality is Locality 0.<br>3: Startup Locality is Locality 3.<br>Other values: reserved.<br>Note: The startup Locality can only be Locality 0 or Locality 3. In the case where an HCRTM event did not occurr, this event may not be used. |

#### 9.4.5.4 Vendor Specific Events

**Start of informative comment**

PC OEM's may define their own events for PCR[6]. Some events may use the event type EV_NO_ACTION. The events will be present in the event log, but won't extend a PCR. PC OEM's may also define events that extend a PCR.

**End of informative comment**

Platform specific events that extend the PCR MUST use the event type EV_COMPACT_HASH. Each event that extends PCR[6] SHALL be logged with the following structure, where event data is a unique string pre-pended with the platform OEM name:

1. TCG_ PCR_EVENT2.pcrIndex = 6
2. TCG_ PCR_EVENT2.eventType = 0Ch (the value for the EV_COMPACT_HASH event type from the table in Section 11.3.1)
3. TCG_ PCR_EVENT2.digests = digest which was extended to the PCR (for all PCR banks)
4. TCG_ PCR_EVENT2.event[] = "<Vendor Name> <Unique identifier>"

 

# 10 Platform Hierarchy (Physical Presence)

**Start of informative comment**

TPM 2.0 augments the concept of Physical Presence with the Platform Hierarchy authorization. The majority of PC Client platforms implemented with TPM 1.2 utilized command-based Physical Presence, using the command TSC_PhysicalPresence, instead of a physical button. Because the platform hierarchy is the point of control for the state of the TPM, it is important that the platform hierarchy be properly protected.

**End of informative comment**

Platform Firmware MUST protect access to the Platform Hierarchy and prevent access to the platform hierarchy by non-manufacturer-controlled components.

1. Platform Firmware SHALL:
    a. Disable the platform Hierarchy, or
    b. If the Platform Hierarchy is enabled:
        i. If the TPM is hidden, Platform Firmware MUST change platformAuth to a high entropy value prior to disabling the TPM as defined in Section 6 TPM Discoverability.
        ii. If the TPM is visible, Platform Firmware MUST change platformAuth to a high entropy value prior to executing 3rd party code, e.g. non-manufacturer controlled UEFI applications.

# 11 Predictive Event Logs

**Start of informative comment**

With the advent of Hash algorithm agility within TPM 2.0, it is possible for a TPM to be configured with one (or more) Hash algorithm at the beginning of a platform's service life and have the Hash algorithm be changed at a later point in the platform's service life. If the OS present environment is using the TPM to protect secrets, it is desirable to have a way to convert TPM protected secrets from the Hash algorithm and PCR allocation with which they were instantiated to the new Hash algorithm and PCR configuration without have to unseal the secrets from the TPM. To support this, the TCG PC Client Physical Presence Interface Specification version 1.3 revision 52 introduced optional PPI operation 33 (LogAllDigests) to support predictive event logs. If the Platform Firmware supports predictive event logs, the requirements in this section must be implemented as defined. As an example, an OS that wants to change the PCR allocation and update its sealed objects will send a PPI operation 33 to Platform Firmware and reboot the system. After user confirmation, Platform Firmware will log digests for all supported hash algorithms and boot to the OS. The OS will receive all the digests for all events and migrate its objects to the desired hash algorithm. Following this, the OS will send a PPI operation to change the PCR allocation to the desired state. After the Platform Firmware performs this second PPI operation, it will produce the event log with the digests for the allocated PCRs.

It is likely, but not required, that Platform Firmware will have internal flags to track this state, as the requirement to calculate the predicative Event Log will only be necessary for one boot cycle (within which the OS present environment will recalculate and reseal its secrets).

**End of informative comment**

If predictive event logs are supported, Platform Firmware SHALL implement all of the following requirements:

1. Platform Firmware MUST support the TCG Physical Presence Operation 33 (LogAllDigests).
2. Platform Firmware MUST measure and record all mandatory events for the currently active PCR banks.
3. Platform Firmware MAY measure and record optional events for the currently active PCR banks.
4. On only the first reboot following receipt of PPI Operation 33 (LogAllDigests), Platform Firmware MUST:
    a. Calculate the tagged Hash for all supported algorithms for all events measured and recording in steps 1 and 2.
    b. Construct the correctly formatted Event Log and record all events with the tagged Hash from step 3a.

# 12 Supporting TCG Opal SSC Block SID enabled devices

**Start of informative comment**

The TCG Storage-Feature Set Block SID Authentication Specification defines a mechanism by which a host application can alert a storage device to block attempts to authenticate the SID authority until a subsequent device power cycle occurs.

This mechanism may be used by Platform Firmware to prevent malicious entities from taking ownership of a TCG Opal storage device which has not been owned.

The TCG PC Client Physical Presence Interface Specification version 1.3 revision 52 introduced PPI operations 96-101 as a convenience to provide a standard way for OS-present applications which can manage a TCG Opal storage device to interface with Platform Firmware through the PPI interface. These operations are optional, but if implemented require corresponding functionality within the Platform Firmware to respond to these requests.

**End of informative comment**

If a platform supports PPI operations 96-101, Platform Firmware MUST implement support for the TCG Storage-Feature Set Block SID Authentication Specification.

# Annex A: TPM Interrupt ASL Example

**Start of informative comment**

This is an informative example of enabling TPM Interrupts.

```
//
    // Operational region for TPM access
    //
    OperationRegion (TPMR, SystemMemory, 0xfed40000, 0x5000)
    Field (TPMR, AnyAcc, NoLock, Preserve)
    {
      ACC0, 8,  // TPM_ACCESS_0
      Offset(0x8),
      INTE, 32, // TPM_INT_ENABLE_0
      INTV, 8,  // TPM_INT_VECTOR_0
      Offset(0x10),
      INTS, 32, // TPM_INT_STATUS_0
      INTF, 32, // TPM_INTF_CAPABILITY_0
      STS0, 32, // TPM_STS_0
      Offset(0x24),
      FIFO, 32, // TPM_DATA_FIFO_0
      Offset(0x30),
      TID0, 32, // TPM_INTERFACE_ID_0
              // ignore the rest
    }

    //
    // Operational region for TPM support, TPM Physical Presence and TPM Memory Clear
    // Region Offset 0xFFFF0000 and Length 0xF0 will be fixed in C code.
    //
    OperationRegion (TNVS, SystemMemory, 0xFFFF0000, 0xF0)
    Field (TNVS, AnyAcc, NoLock, Preserve)
    {
      IRQN,   32, //   IRQ Number for _CRS
      SFRB,   8   //   Is shortformed Pkglength for resource buffer
    }
```

 

```
    //
    // Possible resource settings returned by  _PRS method
    //   RESS : ResourceTemplate with PkgLength <=63
    //   RESL : ResourceTemplate with PkgLength > 63
    //
    // The format of the data has to follow the same format as
    // _CRS (according to ACPI spec).
    //
    Name (RESS, ResourceTemplate() {
      Memory32Fixed (ReadWrite, 0xfed40000, 0x5000)
      Interrupt(ResourceConsumer, Level, ActiveLow, Shared, , , ) {1,2,3,4,5,6,7,8,9,10}
    })

    Name (RESL, ResourceTemplate() {
      Memory32Fixed (ReadWrite, 0xfed40000, 0x5000)
      Interrupt(ResourceConsumer, Level, ActiveLow, Shared, , , )
{1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}
    })

    //
    // Current resource settings for _CRS method
    //
    Name(RES0, ResourceTemplate () {
      Memory32Fixed (ReadWrite, 0xfed40000, 0x5000, REG0)
      Interrupt(ResourceConsumer, Level, ActiveLow, Shared, , , INTR) {12}
    })

    Name(RES1, ResourceTemplate () {
      Memory32Fixed (ReadWrite, 0xfed40000, 0x5000, REG1)
    })


    //
```

```
// Return the resource consumed by TPM device.
//
Method(_CRS,0,Serialized)
{
  //
  // IRQNum = 0 means disable IRQ support
  //
  If (LEqual(IRQN, 0)) {
    Return (RES1)
  }
  Else
  {
    CreateDWordField(RES0, ^INTR._INT, LIRQ)
    Store(IRQN, LIRQ)
    Return (RES0)
  }
}

//
// Set resources consumed by the TPM device. This is used to
// assign an interrupt number to the device. The input byte stream
// has to be the same as returned by _CRS (according to ACPI spec).
//
// Platform may choose to override this function with specific interrupt
// programing logic to replace FIFO/TIS SIRQ registers programing
//
Method(_SRS,1,Serialized)
{
  //
  // Do not configure Interrupt if IRQ Num is configured 0 by default
  //
  If (LEqual(IRQN, 0)) {
    Return (0)
  }
```

```
//
// Update resource descriptor
// Use the field name to identify the offsets in the argument
// buffer and RES0 buffer.
//
CreateDWordField(Arg0, ^INTR._INT, IRQ0)
CreateDWordField(RES0, ^INTR._INT, LIRQ)
Store(IRQ0, LIRQ)
Store(IRQ0, IRQN)

CreateBitField(Arg0, ^INTR._HE, ITRG)
CreateBitField(RES0, ^INTR._HE, LTRG)
Store(ITRG, LTRG)

CreateBitField(Arg0, ^INTR._LL, ILVL)
CreateBitField(RES0, ^INTR._LL, LLVL)
Store(ILVL, LLVL)

//
// Update TPM FIFO PTP/TIS interface only, identified by TPM_INTERFACE_ID_x lowest
// nibble.
// 0000 - FIFO interface as defined in PTP for TPM 2.0 is active
// 1111 - FIFO interface as defined in TIS1.3 is active
//
If (LOr(LEqual (And (TID0, 0x0F), 0x00), LEqual (And (TID0, 0x0F), 0x0F))) {
  //
  // If FIFO interface, interrupt vector register is
  // available. TCG PTP specification allows only
  // values 1..15 in this field. For other interrupts
  // the field should stay 0.
  //
  If (LLess (IRQ0, 16)) {
    Store (And(IRQ0, 0xF), INTV)
```

 

```
    }
    //
    // Interrupt enable register (TPM_INT_ENABLE_x) bits 3:4
    // contains settings for interrupt polarity.
    // The other bits of the byte enable individual interrupts.
    // They should be all be zero, but to avoid changing the
    // configuration, the other bits are be preserved.
    // 00 - high level
    // 01 - low level
    // 10 - rising edge
    // 11 - falling edge
    //
    // ACPI spec definitions:
    // _HE: '1' is Edge, '0' is Level
    // _LL: '1' is ActiveHigh, '0' is ActiveLow (inverted from TCG spec)
    //
    If (LEqual (ITRG, 1)) {
      Or(INTE, 0x00000010, INTE)
    } Else {
      And(INTE, 0xFFFFFFEF, INTE)
    }
    if (LEqual (ILVL, 0)) {
      Or(INTE, 0x00000008, INTE)
    } Else {
      And(INTE, 0xFFFFFFF7, INTE)
    }
  }
}

Method(_PRS,0,Serialized)
{
  //
  // IRQNum = 0 means disable IRQ support
  //
```

 

```
      If (LEqual(IRQN, 0)) {
        Return (RES1)
      } ElseIf(LEqual(SFRB, 0)) {
        //
        // Long format. Possible resources PkgLength > 63
        //
        Return (RESL)
      } Else {
        //
        // Short format. Possible resources PkgLength <=63
        //
        Return (RESS)
      }
    }
```

**End of informative comment**