TCG PC Specific Implementation Specification

Version 1.1
August 18, 2003

Previously published by Trusted Computing Platform Alliance under the title: "TCPA PC Specific Implementation Specification, Version 1.00, September 09, 2001".

Change History

Version	Date	Description
1.00 RC1	August 16, 2001	Proposed initial release candidate
1.00 RC1.1	August 23, 2001	Editorial Corrections.
1.00 RC 2	August 23, 2001	Re-added Preface to give a location for post release comments. Change 7.2.3 EV_ACTION Event Types event: “Booting BCV Device s” Made clarifications to wording in section 8.3.2
1.00 Final	September 09, 2001	Final Release
1.1	July 16, 2003	Moved to a TCG specification. Minor textual corrections.
1.1 RC2	July 23, 2003	Note: Previous version released as RC1 but not labeled as such. Added flag to pre-boot PCR in the case where no measurement can be taken. Changed name of event structure and make reference to TSS spec Make clarifications and corrections to the parameter block definitions in section 8. Added ending pointer to Event Log in StatusCheck.
1.1 RC2A	Aug 13, 2003	Make changes to section 8.2 to clarify flat model. Add TPM Configuration port definition to header
1.1 RC3	Aug 18, 2003	Changed references back to the TCPA Main Specification. Clarifications added to TCG_StatusCheck Added Clarification to LogEventType in TCG_LogEvent Enhanced section 8.2.1 to further describe the behavior of segments. Added TPM and channel error responses. Added configuration port.

Table of Contents

Corrections and Comments

Please send corrections and comments regarding this specification to:

http://www.trustedcomputinggroup.org

1. Introduction and Concepts

Start of informative comment:

The Trusted Computing Group's TCPA Main Specification v1.1b (http://www.trustedcomputinggroup.org) (hereafter, referred to as the TCPA Main Specification) has been written as a platform independent document to enhance trust on computing platforms. As such, the TCPA Main Specification is general in specifying both hardware and software requirements. The goal of the TCG member companies is to ensure compatibility among implementations within each computing architecture. It is expected that companion implementation documents will be created for each architecture.

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

- Usage of PCR registers in the Pre-Boot state through the transition to Post-Boot state.
- How the BIOS, or a component thereof, functions as the Core Root of Trust for Measurement (CRTM).
- Programmatic Interfaces to the BIOS as it performs the functions of the TCG Subsystem (TSS and access to the TPM)
- Behavior entering, during, and exiting power and initialization states.
- Guidelines for Option ROMS.

This specification is based on the *TCPA Main Specification*. 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 TCPA Main Specification will not be defined in this document. If there is a conflict in interpretation between this and the TCPA Main Specification, the concept or functional description as defined in the TCPA Main Specification will take precedence.

This specification also references the following specifications. The reader is expected to be familiar with the concepts and terminology contained in each where relevant:

- Plug and Play BIOS Specification Version 1.0A
- Advanced Configuration and Power Interface Specification; Revision 2.0 July 27, 2000
- BIOS Boot Specification; Version 1.01 January 11, 1996
- Boot Integrity Services Application Programming Interface; Version 1.0
- System Management BIOS Reference Specification
- “El Torito” Bootable CD-ROM Format Specification; Version 1.0; January 25, 1995
- Preboot Execution Environment (PXE) Specification; Version 2.1
- PARTIES (Protected Area Run Time Interface Extension Services) Working Draft;T13 D1367; Revision 3 September 30, 2000.

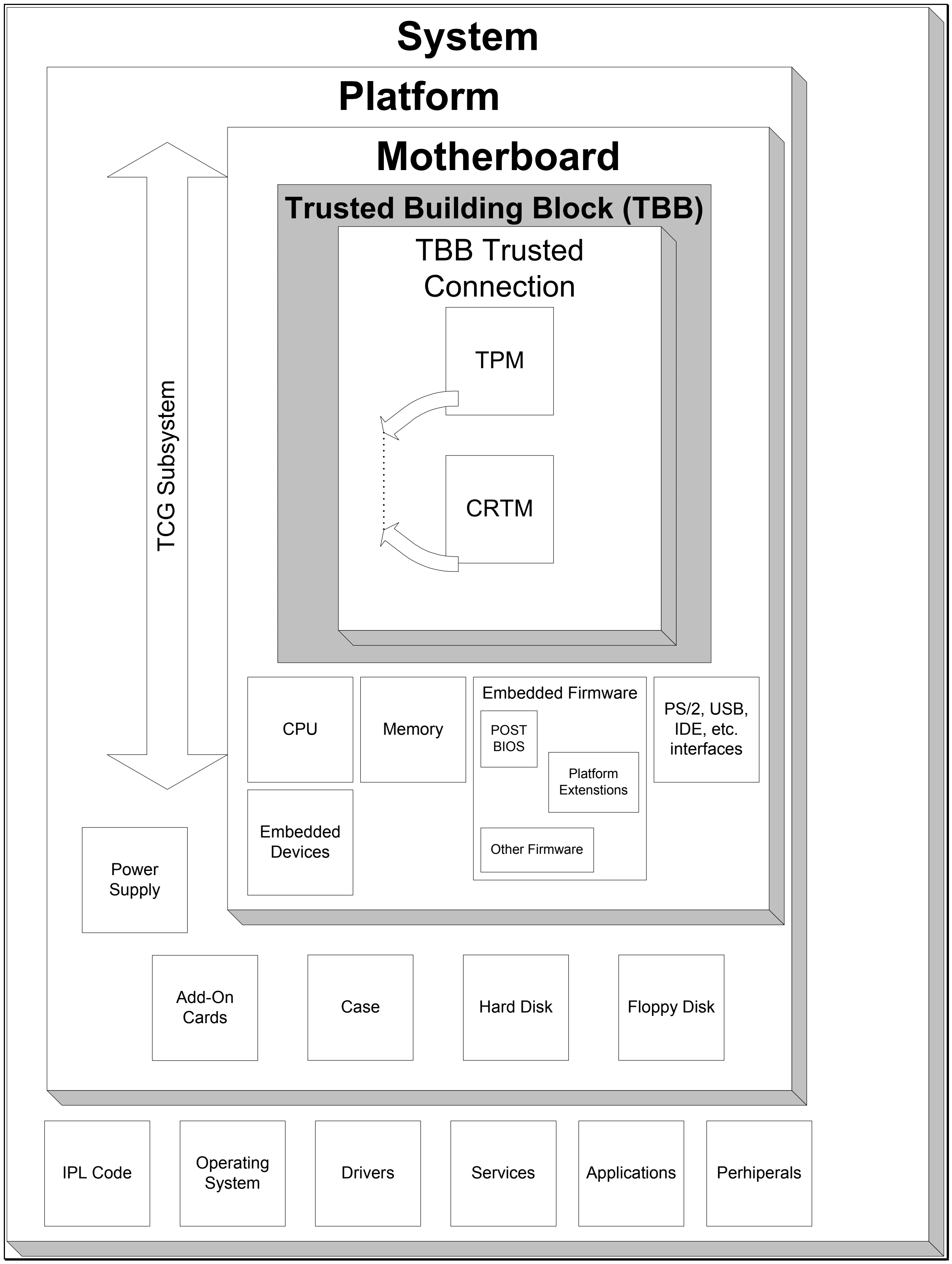
End of informative comment.

1.1 PC Architecture

Start of informative comment:

The concepts and descriptions of the PC architecture are described below in both the diagram and the descriptions. While the diagram infers physical connections, the connections and associations between the components are logical.

End of informative comment.



■ Figure 1-1 Components of the PC

1.2 Definitions

Start of informative comment:

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

End of informative comment.

1.2.1 BIOS Recovery Mode

Start of informative comment:

This is a failure-recovery mode of the BIOS that is invoked by the BIOS Boot Block typically when the main BIOS is corrupt. See section 4.1 BIOS Recovery Mode.

End of informative comment.

1.2.2 Core RTM (CRTM)

Start of informative comment:

The executable component of the RTM that gains control of the Platform upon a Platform Reset. See more detailed description of this in Section 1.3.4 Core RTM (CRTM).

End of informative comment.

1.2.3 Central Processing Unit (CPU)

Start of informative comment:

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

End of informative comment.

1.2.4 Immutable

Start of informative comment:

See Section 1.3.1 Immutable.

End of informative comment.

1.2.5 Initial Program Loader (IPL)

Start of informative comment:

This is the code that executes during the Post-Boot state. The purpose of this code is to load the Post-Boot environment.

End of informative comment.

1.2.6 Manufacturer

Start of informative comment:

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

End of informative comment.

1.2.7 Measurement and Measure

Start of informative comment:

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

End of informative comment.

1.2.8 Motherboard

Start of informative comment:

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

End of informative comment.

1.2.9 Pre-Boot State

Start of informative comment:

The state of the system prior to the invocation of the INT 19h or its equivalent.

End of informative comment.

1.2.10 Post-Boot State

Start of informative comment:

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

End of informative comment.

1.2.11 Platform

Start of informative comment:

The entity that presents and receives information to and from the user. The Platform is composed of the Motherboard to which the CPU and primary peripheral devices are attached[1], the CPU(s), all BIOSes, the TPM, and peripherals attached to the main board.

End of informative comment.

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

1.2.12 Platform Reset

Start of informative comment:

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

End of informative comment.

The event that causes the components of the Platform to enter their reset condition including the TPM (caused by a TPM_Init). Upon a Platform Reset, the CPU MUST begin execution at the CRTM. This event MUST cause a PCI_Reset. Unless otherwise stated, the result of a Platform Reset MUST cause the equivalent of transitioning the motherboard from the S5 state (i.e., It may not cause a transition from S3.)

1.2.13 System

Start of informative comment:

This includes the Platform and all the Post-Boot components that comprise the entire entity that performs actions for, or acts on behalf of, the user.

End of informative comment.

1.2.14 TCPA Main Specification v1.1b

Start of informative comment:

Refers to the TCPA Main Specification v1.1b as released by the Trusted Computing Group. This “TCG PC Specific Implementation Specification” is based on the TCPA Main Specification v1.1b.

End of informative comment.

1.2.15 TCG TSS Specification

Start of informative comment:

Refers to the TCG TSS Specification version 1.1 as released.

End of informative comment.

1.2.16 Trusted Building Block (TBB)

Start of informative comment:

The combination of the CRTM, TPM, connection of the CRTM to the motherboard, and the connection of the TPM to the motherboard. See section 1.3.2 Trusted Building Block (TBB).

End of informative comment.

1.3 Concepts

1.3.1 Immutable

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

1.3.2 Trusted Building Block (TBB)

The combination of the CRTM, TPM, connection of the CRTM to the motherboard, and the connection of the TPM to the motherboard. The connection of the CRTM to the TPM is done through transitive trust of the CRTM connection and the TPM connection.

Since the CRTM and the TPM are the only trusted components of the Motherboard and since indication of physical presence requires a trusted mechanism to be activated by the platform owner, the indication of physical presence MUST be contained within the TBB.

1.3.3 Platform Reset Types

Start of informative comment:

A Cold Boot Platform Reset occurs when transitioning the Platform from a full Power-Off state in which no OS specific state or status is preserved on the 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 which can occur after the Cold Boot Reset from an OS present state.

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

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

End of informative comment.

For all types of Platform Resets the CPU SHALL begin executing code with the CRTM’s Platform initialization code. The Platform MUST perform a Platform Reset. No System component SHALL block the PCI_Reset signal to any of the System components.

1.3.4 Core RTM (CRTM)

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

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

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

- CRTM is the BIOS Boot Block.

  Start of informative comment:

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

End of informative comment.

The Manufacturer MUST control the update, modification, and maintenance of the BIOS Boot Block component, while either the Manufacturer or a 3rd party supplier may update, modify, or maintain the POST BIOS component. If there are multiple execution points for the BIOS Boot Block, they must all be within the CRTM.

- CRTM is the entire BIOS

Start of informative comment:

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

End of informative comment.

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

1.3.5 Boot State Transition

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

1.3.6 Establishing the Chain of Trust

1.3.6.1 Bindings

1.3.6.1.1 Bindings between an Endorsement Key, a TPM, and a Platform.

The relationship between the Endorsement Key, a TPM, and a Platform is described in Section 2.2 of the TCPA Main Specification.

1.3.6.1.2 Binding Methods.

Start of informative comment:

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

Examples:

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

2. The TPM is a SmartCard with the Endorsement Key inside. The SmartCard is on a "common" bus (e.g., USB). The Endorsement Key, as in example 1, is physically bound to the TPM but the TPM now must be bound to the Platform via some cryptographic method. An example is a common shared secret within both the Platform and the SmartCard would enforce each to function only with the appropriate component.

End of informative comment.

2. Integrity Collection & Reporting

2.1 Concepts

2.1.1 Initial TBB control and Platform Reset

Upon Platform Reset the CRTM MUST have control of the TBB.

2.1.2 Transferring Control

Prior to transferring control an executing entity MUST measure the entity to which it will transfer control.

2.2 PCR Usage

Start of informative comment:

This section defines the PCR assignments used for boot time integrity metrics and the methodology for collecting the metrics. The first eight PCRs are defined for use within the Pre-Boot environment (PCR[4] being transition code which is partially Pre-Boot.) Throughout the BIOS boot process a log of all executable code 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.

End of informative comment..

Summary of the defined PCR usage:

PCR Index	PCR Usage
0	CRTM, BIOS and Platform Extensions
1	Platform Configuration.
2	Option ROM Code.
3	Option ROM Configuration and Data.
4	IPL Code (usually the MBR)
5	IPL Code Configuration and Data (for use by the IPL code)
6	State Transition and Wake Events
7	Reserved for future usage. Do not use.

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

Start of informative comment:

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

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

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

End of informative comment.

Entities that MUST be Measured:

- The CRTM's version identifier.
- All firmware physically bound to the motherboard
  - Manufacturer Controlled Embedded Option ROMs

    These are Embedded Option ROMs whose release and update is controlled by the Manufacturer.
  - Embedded SMM code and the code that sets it up.
- ACPI flash data prior to any modifications.
- BIS code (excluding the BIS certificate).

If the measurement of the CRTM, POST BIOS and Embedded Option ROMs cannot be made, the CRTM MUST extend the value 01h to each PCR in the range 0 – 7.

Entities that MAY be Measured:

- **Any other code or information that is relevant to the CRTM, POST BIOS or Platform Extensions.**

Method for Measurement for a Compound BIOS:

The CRTM performs these measurements as follows:

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

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

Method for Measurement for an Integrated BIOS:

The CRTM performs these measurements as follows:

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

2.2.2 PCR[1] - Motherboard Configuration

Start of informative comment:

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

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

The method and policy of disabling measurements is manufacturer specified.

End of informative comment.

These measurements occur only while in the Pre-Boot state.

Entities that MUST be Measured:

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

- If the BIOS loads a CPU microcode update, it is measured.
- Platform Configuration including the state of any disable flags affecting the measurement of entities into this PCR.

Entities that MAY be Measured:

The following entities MUST be measured if measurement of the following entities is enabled by the system. These MAY be Disabled:

- BIS certificate.
- POST BIOS-Based ROM strings.

Entities that MAY be Measured

While the code to implement the above entities is mandatory, the code to implement measurement of these entities is optional. It is not required to measure the components of the following that contain privacy information but if implemented, the rest of the information MUST be.

- ESCD, CMOS and other NVRAM data
- SMBIOS structures
- Passwords

Entities that MUST NOT be Measured

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

Method for Measurement:

The BIOS performs these measurements as follows:

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

2.2.3 PCR[2] - Option ROM Code

Start of informative comment:

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

Visible Portion

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

Hidden Option ROM Code

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

End of informative comment.

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

Entities to be Measured:

- The portion of the Option ROM that is visible to the BIOS.
- The portion of the Option ROM that is not visible to the BIOS is measured by the Option ROM.
- Non-Manufacturer Controlled Embedded Option ROMs

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

Method for Measurement:

The BIOS performs these measurements as follows:

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

Option ROMs perform these measurements as follows when they execute:

1. Measure the event “Un-hiding Option ROM Code” when un-hiding Option ROM code.
2. Measure the “hidden” Option ROM Code.

2.2.4 PCR[3] – Option ROM Configuration and Data

Start of informative comment:

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

End of informative comment.

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

Entities to be Measured:

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

Method for Measurement:

The Option ROM or Application performs these measurements as follows:

1. Measures the event OptionROMConfig.

2. Measure any of the above in any order while executing.

2.2.5 PCR[4] - IPL Code

Start of informative comment:

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

End of informative comment.

Entities to be Measured:

- Each IPL that is attempted and executed.
- Additional code that is loaded by the IPL.

Entities to Exclude:

- Portions of IPL pertaining to the specific configuration of the platform. (e.g., disk geometry in the MBR).

Method for Measurement:

See section 6.2.3 Logging of Boot Events for further detail.

The BIOS performs these steps as follows:

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

A complete description of the method for measuring is found in Section 6 IPL Code, Power States, and Transitions

2.2.6 PCR[5] – IPL Configuration and Data

Start of informative comment:

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

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

End of informative comment.

Entities to be Measured:

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

Method for Measurement:

The IPL code measures all relevant IPL configuration data per its defined events.

The BIOS measures the static data as events defined in Section 7.2.2 Platform Specific Event Log

2.2.7 PCR[6] – State Transition

Start of informative comment:

Events recorded to this PCR are events related to State Transitions and Wake Events.

End of informative comment.

Entities to be Measured:

- Wake Events
- All relevant State Transitions.

Method for Measurement:

Wake events are measured by the Pre-Boot components as defined in Section 7.2.2 Platform Specific Event Log

State Transitions are measured by the Post-Boot components as defined in Section 7.2.2 Platform Specific Event Log

2.2.8 PCR[7] – Reserved

Start of informative comment:

Reserved to future Pre-Boot use.

End of informative comment.

3. Platform Setup and Configuration

3.1 Pre-Boot ROM-based Setup

Upon completion, this setup utility MUST perform a Platform Reset. This includes setup utilities provided by both the motherboard-based BIOS and Option ROMs.

Entry into this state is measured as event “Entering ROM Based Setup”.

3.2 Post-Boot ROM-based Setup

Start of informative comment:

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

End of informative comment.

The setup utility MUST NOT allow changes to platform configuration unless the Post-boot environment can measure the event or the setup utility provides a mechanism to notify the Post-Boot OS that a change occurred.

3.3 Reference Partition

This is treated as IPL code. The setup utility within the reference partition MUST measure events that affect platform configuration.

3.4 OS Based Setup Utility

The setup utility MUST measure events that affect platform configuration.

4. Maintenance

Start of informative comment:

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

End of informative comment.

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

4.1 BIOS Recovery Mode

Start of informative comment:

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

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

- A hardware signal (e.g., GPIO or similar) to “disable” the TPM (until the next TPM_Init); or,
- Modifications to CRTM recovery code which would extend the value 1 to the pre-Boot PCRs; or,
- Disable the TPM by calling TPM_Startup(stTrpe=TPM_ST_DISABLE)

End of informative comment.

It MUST NOT be possible for a BIOS Recovery Mode to allow impersonation of another valid boot state. This applies to the values in the pre-Boot PCRs. Upon completion, the BIOS Recovery Code MUST cause a Platform Reset.

4.2 Flash Maintenance

Start of informative comment:

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

End of informative comment.

4.2.1 Manufacturer Approved Environment (MAE)

Start of informative comment:

This is using a utility that is approved by the Manufacturer of the Platform.

End of informative comment.

The CRTM MAY be updated while in MAE.

4.2.2 Non-Manufacturer Approved Environment (NMAE)

Start of informative comment:

This is using a utility that is not approved by the Manufacturer of the Platform.

End of informative comment.

The CRTM MAY NOT be updated while in NMAE.

5. TCG Credentials

All TCG Credentials MUST be represented as Certificates as defined in Section “9.5 Instantiation of Credentials as Certificates” in the TCPA Main Specification.

5.1 Platform Certificate

Distribution is manufacturer controlled.

5.2 Platform Conformance Certificate

Distribution is manufacturer controlled.

5.3 Method of Verification

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

5.4 Validation Certificate Header

If present, the Validation Certificate will be contained within the Option ROM header as specified below according to the “Plug and Play BIOS Specification”.

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

6. IPL Code, Power States, and Transitions

6.1 Architecture and Definitions

Start of informative comment:

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

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

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

End of informative comment.

6.2 Procedure for Transitioning the TPM from Pre-Boot to Post-Boot

Start of informative comment:

In order to transition from the Pre-Boot state to the Post-Boot state in a TCG protected environment, a number of steps need to be performed. This section of the specification will outline and describe these steps.

End of informative comment.

6.2.1 Extending PCR[4] – The IPL Code

Start of informative comment:

Just before handing control over to the operating system, the BIOS needs to perform several actions in order to assure that trust in the platform has been maintained. One of the important events that need to occur is the extending of PCR[4]. This is done utilizing the BIOS function INT 1Ah where AH = BBh and AL = 01h (HashLogExtendEvent) to hash the first 512 bytes of the boot device.

End of informative comment.

6.2.2 Extending PCR[5] – IPL Configuration and Data

Start of informative comment:

PCR[5] is reserved for any configuration data that various transition code may need. For example, if a BIOS is transitioning to a MBR on a hard drive, then there may be no configuration needed. However, this PCR is to be utilized and extended by any boot loader for variable data.

End of informative comment.

6.2.3 Logging of Boot Events

Start of informative comment:

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

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

- The type of device that was booted to and specific information about this device, which can uniquely identify it within the system.
- Each attempt to a boot device should measure:
  - An EV_SEPARATOR event to all pre-boot PCR (0-7) to delimit pre-boot and post-boot events
  - The boot device attempted
- Any time a boot device returns back to the BIOS through INT 18h should be logged as an event.

End of informative comment.

Prior to calling Int19h, the event EV_SEPARATOR SHALL be measured to the pre-boot PCRs (PCR[0-7]). This SHALL be followed by measuring the event “Calling INT 19h” to PCR[4]. If a boot device returns, an event indicating the nature of the return SHALL be measured to PCR[4]. Subsequent attempts to boot SHALL measure the boot device to PCR[4] and the event EV_SEPARATOR to the pre-boot PCRs (PCR[0-7]).

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

Start of informative comment:

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

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

End of informative comment.

6.2.5 Various Boot Devices and Special Treatment they may receive

Start of informative comment:

- BIOS Aware IPL Devices (BAID):

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

- Legacy IPL Devices

Option ROMs will have already been measured. INT 18h and INT 19h events will be measured as events per Section 7.2.3 EV_ACTION. It is the Option ROM's responsibility to measure any additional code loaded.

- PNP Cards
  - Boot Connection Vector (BCV)

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

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

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

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

- El Torito

  When the BIOS boots to a CDROM device that supports the El Torito specification, it first loads the Booting Catalog. If there is more then one boot image on the CD, the user is then prompted to select the boot image. Using this selection and the Booting Catalog, the BIOS loads the first 512 bytes of the CDROM that contains the image into memory and turns control over to this code. Prior to jumping to this code, the BIOS MUST measure the initial IPL image of the boot image code into PCR[4]. The BIOS code will also measure the entire contents of the boot catalog into PCR[5]. The measurement of the boot catalog will be done prior to the measurement of the initial IPL image.
- Legacy Reboot

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

End of informative comment.

6.3 Power States, Transitions, and TPM Initialization

Start of informative comment:

Suspend is designed to reduce overall power consumption under software control. For instance, Windows 2000 or Linux support a power management standard called ACPI (Advanced Control Power Interface Specification). This standard defines a set of power states that can change the behavior of a device during sleeping states.

End of informative comment.

6.3.1 Definitions and Conditions during Power States

Start of informative comment:

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

Platforms and Post-BIOS Operating Systems MUST support ACPI.

End of informative comment.

6.3.1.1 S1: Stand-by - Low wakeup latency sleeping state

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

6.3.1.2 S2: Stand-by with CPU context lost

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

6.3.1.3 S3: Suspend To Ram

TPM State:	S3 is the most complex mode to handle, because PCR values are to be preserved by the platform during this mode. The mechanism to preserve the values cannot be accessible outside the TPM. During S3 the TPM must prohibit all TPM functions.
Entering S3:	The post-boot driver MAY issue the TPM_SaveState.
During S3 :	May have power. This is hardware design dependent. If the TPM has the ability to preserve the contents of the PCRs without power, no power is needed to the TPM. However, if the TPM cannot maintain the contents of the PCRs without power, the Motherboard MUST provide sufficient power to the TPM to maintain the PCRs.
Exiting S3:	The command to restore the PCRs is issued by the CRTM.

6.3.1.4 S4 OS: Suspend To Disk

TPM State:	All power, including auxiliary, is removed.
Entering S4:	Nothing to do.
During S4:	The TPM is off – Nothing to do.
Exiting S4:	The PCRs will be lost, including the PCRs used by the OS, therefore the OS must establish new integrity. The OS, therefore, cannot attest to its original power-on state.

6.3.1.5 S4 BIOS: Suspend To Disk

TPM State:	All power, including auxiliary, is removed.

Entering S4:	Nothing to do.
During S4:	The TPM is off – Nothing to do.
Exiting S4:	The PCRs will be lost, including the PCRs used by the OS, therefore the OS must establish new integrity. The PCR contents may be different from S4 from OS.

6.3.1.6 S5: Off State

TPM State:	All power, including auxiliary, is removed.
Entering S5:	Nothing to do.
During S5:	The TPM is off – Nothing to do.
Exiting S5:	The PCRs will be lost, including the PCRs used by the OS, therefore the OS must establish new integrity.

6.3.2 Power State Transitions

Start of informative comment:

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

End of informative comment.

In the following pseudo code is a suggested set of implementation that generalized the control flow of the motherboard during the pre-Boot state. Not all conditions and error states are included. This intended only as a guide.

6.3.2.1 S5 → S0

Start of informative comment:

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

End of informative comment.

Starting from a power off state.

```
   MAInitTPM (stType = TPM_ST_CLEAR)
   if (MAInitTPM returned OK)
   {
      MAHashAllExtendTPM(CRTM version, PCR[0])
   }
   else  // MAInitTPM returned Error
MAInitError:
   {
      if (PMInitCRTM() indicated TPM failure)
      {
         // Keep communication path open.
         GoTo POST_BIOS    // Transfer control to POST BIOS.
      }
      else // Assume communication path failed
      {
         if (Disable TPM Interface is provided)
         {
            Disable Interface to TPM
         }
         else
         {
            Disable the platform
         }
      }
   }
   if (Normal boot)
   {
      MAHashAllExtendTPM(Initial POST BIOS, PCR[0])
      GoTo POST_BIOS    // Transfer control to POST_BIOS
   }
   // Note: the following else cluase is optional depending if either the
   // BIOS Recovery Mode or a Utility requiring physical presence
   // indication from the boot state is part of the motherboard's design.
   else if (executing BIOS Recovery Mode)
   {
      MAHashAllExtendTPM(BIOS Recovery Code, PCR[0])
      GoTo BIOS_Recovery_Code
   }
   else if (indication of physical presence given to BIOS)
   {
      if (Platform requires physical presence during
            boot state)
      {
         MAHashAllExtendTPM(Utility, PCR[0])
         MAPhysicalPresenceTPM(  TPM_PC_PHYSICAL_PRESENCE_MASK_SW |
                                 TPM_PC_PHYSICAL_PRESENCE_PRESENT)
         GoTo Physical_Presence_Utility
      }
   }

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

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

   TCG_HashLogExtendEvent(Platform Configuration, PCR[1])

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

INT_18:
   Choose next IPL Code
   TPMHashAllExtendCRTM(PCR[4], Chosen IPL Code)
   TPMHashAllExtendCRTM(PCR[0-7], EV_Separator)
   TPMHashAllExtendCRTM(PCR[4], "Calling INT 19h")
   INT 19h // To Execute IPL Code

IPL:
   TCG_HashLogExtendEvent(IPL Configuration Data, PCR[5])
   Transfer Control to OS Loader
   if (OS loader fails to load OS)
GoTo INT_18
```

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

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

END
```

6.3.2.2 S1 → S0

Start of informative comment:

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

End of informative comment.

No Action

6.3.2.3 S2 → S0

Start of informative comment:

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

End of informative comment.

No Action

6.3.2.4 S3 → S0

Start of informative comment:

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

The OS must assure prior to entering S3 that the TPM as preserved the required values.

There must be a countermeasure in the event POST is modified by malicious code and the platform resumes from S3 executing that code. After modifying BIOS, the OS is required to transition the platform to S5 before allowing a transition to S3. This is to allow the new BIOS to be measured. The CRTM is responsible for enforcing this behavior.

End of informative comment.

CRTM MUST be able to determine if there has been an update to any portion of the BIOS since the previous transition from S5. If the CRTM detects a modification to BIOS since the last transition from S5, the CRTM MUST either:

- Force the platform to transition to S5, or
- Make the contents of PCR[0] invalid.

```
MAInitTPM (stType = TPM_ST_STATE)
If MAInitTPM returned OK
{
   If BIOS modified since last S5
   {
      Force transition to S5.
      or
      Invalidate PCR[0].
   }
   Transfer control to the OS.
}
else
{
   Force transition to S5.
   GoTo MAInitError in 6.3.2.1 S5 → S0
}
```

6.3.2.5 S4 → S0

Start of informative comment:

This is where the IPL instead of loading the OS loader will load memory form a hard disk. Platform Reset is asserted. The full BIOS initialization sequence is executed just like the S5->S0 transition. If there are any changes to the Platform’s components or configuration, measuring these changes is the responsibility of both the Pre-Boot and the Post-Boot states.

End of informative comment.

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

7. Event Logging

7.1 ACPI Table Usage

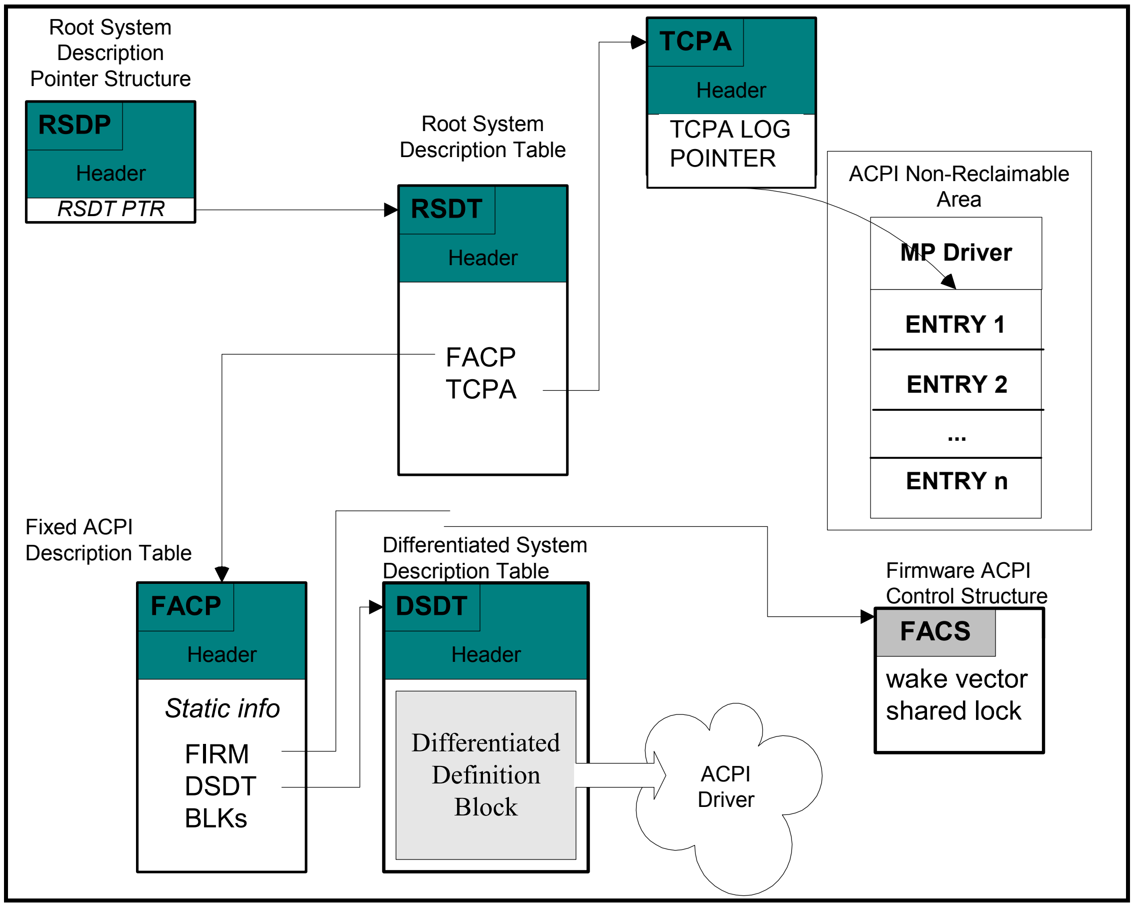
Start of informative comment:

A system's firmware uses an ACPI table to identify the system's TCG capabilities to the Post-Boot environment. The information in this table is not guaranteed to be valid until the BIOS performs the transition from pre-boot state to post-boot state.

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

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

End of informative comment.


Root System Description Pointer Structure
RSDP
Header
RSDT PTR
Root System Description Table
RSDT
Header
FACP
TCPA
TCPA
Header
TCPA LOG POINTER
ACPI Non-Reclaimable Area
MP Driver
ENTRY 1
ENTRY 2
...
ENTRY n
Fixed ACPI Description Table
FACP
Header
Static info
FIRM
DSDT
BLKs
Differentiated System Description Table
DSDT
Header
Differentiated Definition Block
ACPI Driver
Firmware ACPI Control Structure
FACS
wake vector
shared lock

■ Figure 7-1 ACPI Structure

Field	Byte Length	Byte Offset	Description
Header			
Signature	4	0	This is the byte sequence 41h, 50h, 43h, 54h which is the ASCII string 'TCPA' for compatibility with the original specification.
Length	4	4	Length, in bytes, of the entire TCG Table. The length implies the number of Entry fields at the end of the table.
Revision	1	8	1
Checksum	1	9	Entire table must sum to zero.
OEMID	6	10	For instance: “HPINVT”
OEM Table ID	8	16	For the TCG Table, the table ID is the manufacture model ID.
OEM Revision	4	24	OEM revision of TCG table for supplied OEM Table ID.
Creator ID	4	28	Vendor ID of utility that created the table.
Creator Revision	4	32	Revision of utility that created the table.
Reserved	2	36	Reserved for future assignment by this specification, set to 0000h.
Log Area Maximum Length (LAML)	4	38	Identifies the maximum length (in bytes) of the system’s pre-boot TCG event log area. ***Note***: For TCG 1.1b, this maximum log size is 64KB.
Log Area Start Address (LASA)	8	42	Contains the 64-bit physical address of the start of the system's pre-boot TCG event log area, in QWORD format. ***Note***: The log area ranges from address LASA to LASA+(LAML-1).

7.2 Measurement Event Log

Start of informative comment:

TCG_HashLogExtendEvent

Events are logged using the TSS_PCR_EVENT structure which is defined and documented in the TCG TSS Specification These structures are stored as an unstructured array within the ACPI data area as defined in Section 7.1 ACPI Table Usage. None of the pre-Boot entities, including ACPI, are required to interpret this data. The storage of this data using ACPI is a convenience because there are defined mechanisms already in place to allow the transfer of this information to the Post-Boot State. Once the Post-Boot State controls the platform, the Post-Boot OS is expected to read this data and transfer it to its own event log.

End of informative comment.

The instantiation of the event log is an array of TSS_PCR_EVENT structures as defined below.

7.2.1 Platform Independent Event Log Structure

Platform independent events SHALL be done using the events identified in the TCPA Main Specification. Examples of these are Validation Certificates. These are logged using the EV_CODE_CERT event type.

7.2.2 Platform Specific Event Log

For the events described in this section the EventType SHALL be EV_PLATFORM_SPECIFIC and the event field within the TSS_PCR_EVENT structure SHALL be the PlatformSpecificEventLogStruct as defined in Section 7.2.2.1 Platform Specific Event Log Structure.

7.2.2.1 Platform Specific Event Log Structure

The Events shall be the following structure.

```
PlatformSpecificEventLogStruct      STRUCT
   EventID        DD ?  / Tag as defined in
                           Section 7.2.2.2 Platform Specific Event Tags
   EventDataSize DD  ?  / Size of EventData
   EventData      DB ?  / EventData
PlatformSpecificEventLogStruct      ENDS
```

7.2.2.2 Platform Specific Event Tags

The EventID and EventDataSize elements are represented in big endian format.

7.2.2.2.1 SMBIOS structure

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

EventID = 0001h

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

7.2.2.2.2 BIS Certificate

The BIS Certificate SHALL be logged using the following:

EventID = 0002h

EventData[] = Raw BIS Certificate

7.2.2.2.3 POST BIOS ROM Strings

The BIOS ROM Strings SHALL be logged using the following:

EventID = 0003h

EventData[] = Hash of POST BIOS ROM Strings

7.2.2.2.4 ESCD

The ESCD SHALL be logged using the following:

EventID = 0004h

EventData[] = Hash of ESCD Data

7.2.2.2.5 CMOS

The CMOS SHALL be logged using the following:

EventID = 0005h

EventData[] = Raw CMOS Data

7.2.2.2.6 NVRAM

The NVRAM SHALL be logged using the following:

EventID = 0006h

EventData[] = Raw NVRAM contents

7.2.2.2.7 Option ROM Execute

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

EventID = 0007h

EventData[] = OptionROMExecuteStructure(including the PFA)

7.2.2.2.8 Option ROM Configuration

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

EventID = 0008h

Event[] = OptionROMConfigStructure(include PFA)

7.2.2.2.9 Option ROM Microcode Update

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

EventID = 000Ah

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

7.2.3 EV_ACTION Event Types

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

EventType = EV_ACTION

Event[] = ASCII string of the following:

String	Purpose and Comments	PCR
"Calling INT 19h"	BIOS is calling INT 19h. If no additional strings posted in log that means that the software which 'hooked' the INT 19 vector did not return control to the BIOS.	4
"Returned INT 19h"	BIOS Received control back from prior INT19h invocation. If the called code is not TCG-aware it may have loaded additional unmeasured code. However there is a log entry showing entry to (and measurement of) untrusted code.	4
"Return via INT 18h"	BIOS Received control back via INT 18h If the called code is not TCG-aware it may have loaded additional unmeasured code. However there is a log entry showing entry to (and measurement of) untrusted code.	4
"Booting BCV Device s"	BIOS is IPL/Booting a BCV Device. The value 's' is a ASCII string that unambiguous describes the boot device. This SHOULD include an indication of logical or physical device location and any ID string returned by the device.	4
"Booting BEV Device s"	BIOS is IPL/Booting a BEV Device. The value 's' is an ASCII string supplied by the BEV device.	4
"Entering ROM Based Setup"	BIOS is entering ROM based Setup during pre-boot environment.	0
"Booting to Parties N"	BIOS is IPL/Booting from a Parties Partition #N. The value n is the actual numeric value of the partition number represented as a printable ASCII hex value. (e.g. partition zero would get the string value "0"). Where N is the index into the BEER table.	4
"User Password Entered"	User has entered the correct user password.	4
"Administrator Password Entered"	User has entered the correct administrator password.	4
"Password Failure"	The typed password did not match the stored password after a specified number of retries.	4

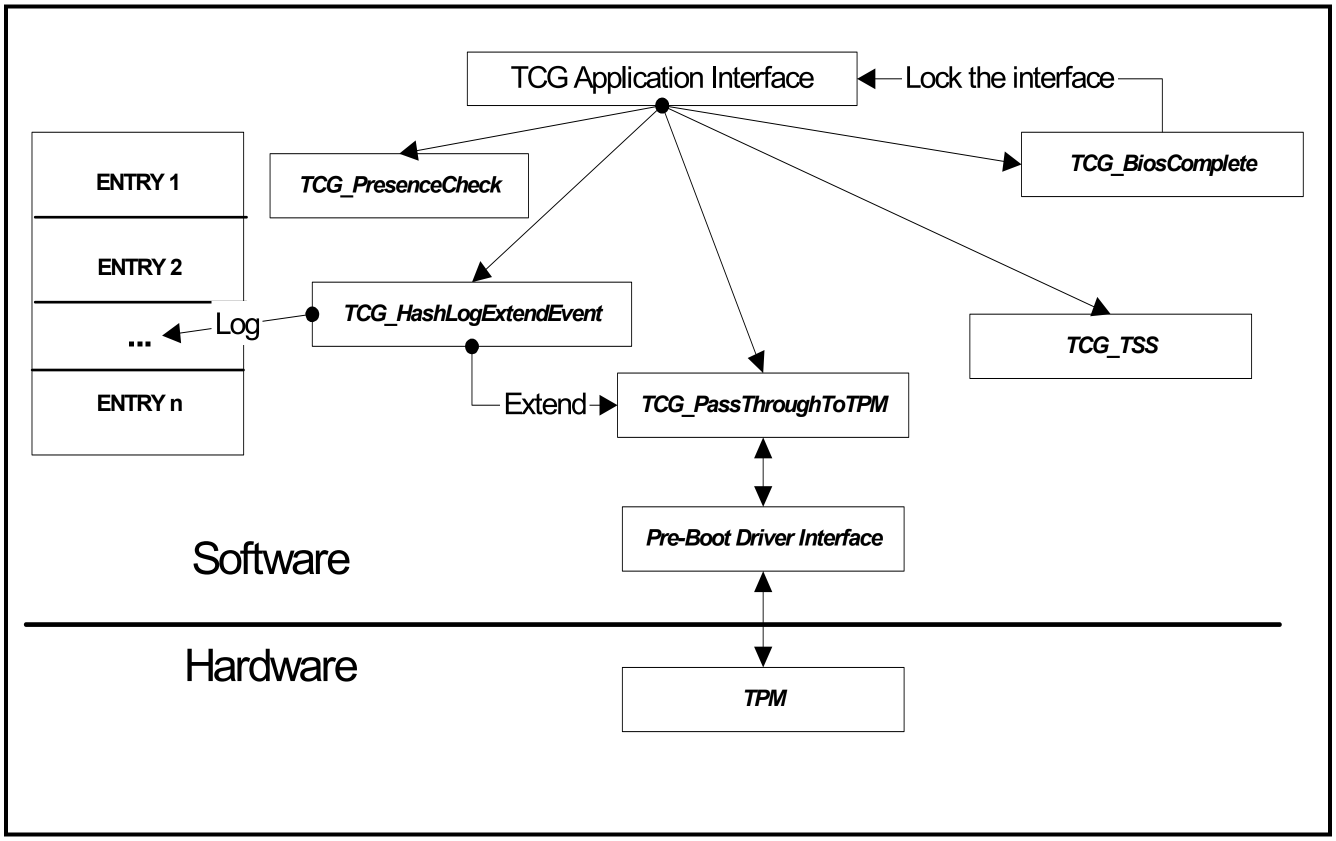
“Wake Event n”	Cause of the power to be applied to the platform where n is the WfM wake source (e.g. wake source zero would get the string value “0”).	1
“Boot Sequence User Intervention”	User altered the boot sequence	
“Chassis Intrusion”	The case was opened.	1
“Non Fatal Error”	A non-fatal POST error (e.g. keyboard failure) was encountered. This is any error that allows the system to continue the boot process	1
“Start Option ROM Scan”	BIOS has started the Option ROM scan.	2
“Unhiding Option ROM Code”	Unhiding Option ROM Code	2
“<OpRom Specific non-IPL String>”	An Option ROM vendor specific string for non-Boot/IPL events.	3
“<OpRom Specific IPL String>”	An Option ROM vendor specific string for Boot/IPL events.	5

8. Implementation

Start of informative comment:

The TCG interfaces in the Pre-Boot environment are: TCG Application Interface, TPCA Pre-Boot Driver Interface and a set of ACPI structure definitions containing the Event Log. A diagram of these interfaces and their relationship is in Figure 8-1 Pre-Boot Interfaces below:

End of informative comment.



■ Figure 8-1 Pre-Boot Interfaces

8.1 Application Level Interface

Start of informative comment:

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

Pre-Boot entities may intercept the INT 1Ah interface for the purpose of adding TSS functions. No mechanism is required nor suggested for preventing an attack by intercepting the INT 1Ah

End of informative comment.

Entering this interface the CPU MUST be in either real-mode or big-real-mode, and the gate A20 state is undefined. Upon exit the interface and gate A20 state MUST return in the same mode.

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

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

8.1.1 General Calling Convention

Each function below will have the following general calling convention:

On entry:

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

On return:

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

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

8.1.2 Return Codes

The following are the defined error codes the pre-Boot functions MAY return:

Return Code	Value	Description
TCG_PC_OK	0000h	The function returned successful.
TCG_PC_TPMERROR	TCG_PC_OK + 01h | (TPM driver error << 16)	The TPM driver returned an error. The upper 16 bits contain the actual error code returned by the driver as defined in Section 8.2.3.6 Error and Return Codes.
TCG_PC_LOGOVERFLOW	TCG_PC_OK + 02h	There is insufficient memory to create the log entry.
TCG_PC_UNSUPPORTED	TCG_PC_OK + 03h	The requested function is not supported.

8.1.3 Parameter Block

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

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

8.1.4 TCG_StatusCheck

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

This function call verifies the presence of the TCG BIOS interface and provides the caller with the version of this specification to which the implementation complies. If required, MPInitTPM MAY be called to initialize the MP Driver during the first invocation of this function..

On entry:

(AH)	=	BBh
(AL)	=	00h

On return:

(EAX)	=	Return code. Set to 00000000h if the system supports the TCG BIOS calls.
(EBX)	=	'41504354h
(CH)	=	TCG BIOS Major Version (01h for version 1.0)
(CL)	=	TCG BIOS Minor Version (00h for version 1.0)
(EDX)	=	BIOS TCG Feature Flags (None currently defined. MUST be set to 0)
(ESI)	=	Pointer to the beginning of the Event Log
(EDI)	=	Pointer to the first byte of the last event in the log.

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

8.1.5 TCG_HashLogExtendEvent

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

This function performs the functions of the: TSS_HashAll, TPM_Extend, and TSS_LogEvent operation for the data region specified by the caller. The caller should verify the availability of this function by issuing a previous call to the Presence Check function, that way the caller can be assured that calls to this function preserve the register contents (including the upper 16 bits of 32-bit registers).

On entry:

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

On return:

(EAX)	=	Return Code as defined in Return Codes 8.1.2
(DS:SI)	=	Referenced buffer updated to provide return results.

All other registers are preserved.

8.1.5.1 HashLogExtendEvent Input Parameter Block

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

8.1.5.2 HashLogExtendEvent Output Parameter Block

Offset	Size	Field Name	Description
00h	WORD	OPBLength	The length, in bytes, of the output parameter block. If hash algorithm is SHA-1 which has a 20h byte output this value will be 0028h.
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	DWORD	EventNumber	The event number of the event just logged.
08h	Depends on hash function	HashValue	The TPM_HASH result of the HashAll function. Since the TCPA Main Specification states this hash functions use SHA-1, the size of this field will be 20h bytes.

8.1.6 TCG_PassThroughToTPM

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

This function provides a pass-through capability from the caller to the system's TPM. Refer to the TPM implementation appendix of the TCPA Main Specification for input/output parameter block formats. The caller should verify the availability of this function by issuing a previous call to the Presence Check function, that way the caller can be assured that calls to this function preserve the register contents (including the upper 16 bits of 32-bit registers).

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

On entry:

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

On return:

(EAX)	=	Return Code as defined in Return Codes 8.1.2
(DS:SI)	=	Referenced buffer updated to provide return results.

All other registers are preserved.

8.1.6.1 TPM Input Parameter Block

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

8.1.6.2 TPM Output Parameter Block

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

8.1.7 TCG_ShutdownPreBootInterface

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

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

Calling this function is optional.

On entry:

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

On return:

(EAX)	=	Return Code as defined in Return Codes 8.1.2

All other registers are preserved.

8.1.8 TCG_LogEvent

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

This function MUST provide the TSS capability TSS_LogEvent.

On entry:

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

On return:

(EAX)	=	Return Code as defined in Return Codes 8.1.2
(DS:SI)	=	

All other registers are preserved.

8.1.8.1 LogEvent Input Parameter Block

Offset	Size	Field Name	Description
00h	WORD	IPBLength	The length, in bytes of the input parameter block, set to 001Ch
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	DWORD	HashDataPtr	The 32-bit physical address of the start of the data buffer to be logged.
08h	DWORD	HashDataLen	The length, in bytes, of the buffer referenced by HashDataPtr.
0Ch	DWORD	PCRIndex	The PCR number to which the event is the logged.
10h	DWORD	LogEventType	The EventType code to be logged with the resultant hash, as defined by the TCPA Main Specification. This information is informational only as it is a duplicate of the information provided within the TSS_PCR_EVENT data structure. The BIOS MAY return an error if the value in this parameter does not match the corresponding value within the TSS_PCR_EVENT data structure.
14h	DWORD	LogDataPtr	The 32-bit physical address of the start of the data

Offset	Size	Field Name	Description
			buffer containing the TSS_PCR_EVENT data structure.
18h	DWORD	LogDataLen	The length, in bytes, of the TSS_PCR_EVENT data structure.

8.1.8.2 LogEvent Output Parameter Block

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

8.1.9 TCG_HashAll

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

This function MUST provide the TSS capability: TSS_HashAll.

On entry:

(AH)	=	BBh
(AL)	=	05h
(ES)	=	Segment portion of the pointer to the HashAll input parameter block
(DI)	=	Offset portion of the pointer to the HashAll input parameter block
(DS)	=	Segment portion of the pointer to the Digest
(SI)	=	Offset portion of the pointer to the Digest
(EBX)	=	41504354h
(ECX)	=	0
(EDX)	=	0

On return:

(EAX)	=	Return Code as defined in Return Codes 8.1.2
(DS:SI)	=	Referenced buffer updated to provide return results.

All other registers are preserved.

8.1.9.1 HashAll Input Parameter Block

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

8.1.10 TCG_TSS

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

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

The TSS in and out Operands are defined in the TCG TSS Specification.

On entry:

(AH)	=	BBh
(AL)	=	06h
(ES)	=	Segment portion of the pointer to the TSS input parameter block
(DI)	=	Offset portion of the pointer to the TSS input parameter block
(DS)	=	Segment portion of the pointer to the TSS output parameter block
(SI)	=	Offset portion of the pointer to the TSS output parameter block
(EBX)	=	41504354h
(ECX)	=	0
(EDX)	=	0

On return:

(EAX)	=	Return Code as defined in Return Codes 8.1.2
(DS:SI)	=	Referenced buffer updated to provide return results.

All other registers are preserved.

8.1.10.1 TSS Input Parameter Block

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

8.1.10.2 TSS Output Parameter Block

Offset	Size	Field Name	Description
00h	WORD	OPBLength	The length, in bytes, of the output parameter block, set to 0004h plus the size of TSSOperandOut.

Offset	Size	Field Name	Description
02h	WORD	Reserved	Reserved for future definition by this specification, set to 0000h.
04h	BYTE	TSSOperandOut	The TSS Operand Parameter Block received from the TSS

8.1.11 TCG_BIOSReserved

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

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

8.1.12 TCG_BIOSVendorReserved

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

Reserved for Vendor specific functions.

On entry:

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

8.2 TPM Driver Interfaces

8.2.1 Module Architectures

8.2.1.1 TPM Supplied BIOS Drivers

Start of informative comment:

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

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

End of informative comment.

8.2.1.2 Object Format of BIOS Drivers

Both drivers provide a standard object format to the BIOS vendor as described in this section.

Table 8-1 BIOS Driver Header below describes what the header of the BIOS drivers will look like and where the driver code should start. The BIOS will move the driver into high memory, and then call the start code of the driver. The driver code MUST be relocate-able and MUST be 32-bit code, capable of running in a flat segment memory model with all flat, 32 bit pointers including all data segments.

8.2.1.3 Segments

In the Memory Absent driver, the data segment SHALL be a flat, 32 bit address with EDS set to zero.

In the Memory Present driver, the data segment SHALL be a flat, 32 bit address but EDS may be set either zero or any valid non-zero value.

8.2.1.4 Driver and TPM errors

Start of informative comment:

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

End of informative comment.

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

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

- Table 8-1 BIOS Driver Header

Offset	Size	Default-Value	Description
00h	WORD	55AAh	Signature used to designate the start of the BIOS driver. This is deliberately set different than the Option ROM header.
02h	DWORD		Pointer to beginning of code (Offset to entry point for the driver).
06h	WORD		Total size of the driver in bytes (including the header).
08h	DWORD	00000000h	Base address of the TPM (as set by BIOS).
0Ch	DWORD	00000000h	Optional 2nd base address. This is for memory and I/O mapped or decoding I/O location/address (as set by BIOS).
10h	BYTE	FFh	IRQ Level (00h is not assigned FFh is not required) (as set by BIOS and MUST be sharable).
11h	BYTE	FFh	DMA Channel (FFh in none assigned) (as set by BIOS).
12h	BYTE		XOR-Checksum of entire driver including this header at driver builds time. This is not maintained by the BIOS.
13h	BYTE	00h	Reserved and set to zero.
14h	DWORD	00000000h	PCI PFA if appropriate.
18h	DWORD	00000000h	USB, CardBus, etc
1Ch	DWORD		Location of TPM Configuration port
20h	Variable		Reserved for vendor specific data or is the entry point if vendor specific data not used.
XXh			Entry point into driver.

8.2.1.5 Basic assumptions for both BIOS Drivers

8.2.1.5.1 CMOSTimer

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

8.2.1.5.2 Motherboard Initialization

All Motherboard chipset initialization (concerning the communication channel to TPM device) will be completed by the CRTM or POST-BIOS prior to calling the BIOS **-CRTM-Driver** or **POST-Driver.**

8.2.1.5.3 Basic requirements

The BIOS drivers MUST fulfill the following requirements:

- The drivers MUST be completely self-contained since no BIOS services should be used;
- The drivers MUST check the validity of all the input parameters;

- The drivers MUST include block chaining for the transmission of large data blocks to and from the TPM device;
- The drivers are responsible to add and remove all TPM-Vendor specific protocol information to the TCG-Transfer-Data (TCG-Command);

8.2.2 Memory Absent (MA) Driver

Start of informative comment:

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

It is expected to be used in the BIOS Boot Block of Compound BIOSes. It is not required for Motherboards containing an Integrated BIOS to implement this driver; instead they may choose to implement only the Memory Present (MP) Driver described in Section 8.2.3 Memory Present (MP) Driver

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

End of informative comment.

8.2.2.1 MA Driver Limitations

- No DMA
- No IRQ
- No Physical Memory
- MA-Driver Register usage table (General-Purpose and Segment register):

Register	**Size**	**In / Out**	**Description**
EAX	32	Not available	Driver must preserve this register.
EBX	32	Not available	Driver must preserve this register.
ECX	32	In / Out	Driver I/O; Set by the caller.
EDX	32	In / Out	Driver I/O; Set by the caller.
ESI	32	Not available	Driver must preserve this register.
EDI	32	Not available	Driver must preserve this register.
ESP	32	In (Offset)	Offset of the pointer to argument packet see Section 8.2.2.2. Set by the caller.
SS	16	In (Segment)	Segment of the pointer to argument packet see Section 8.2.2.2. Set by the caller.

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

- Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

8.2.2.2 MA Driver Argument Packet Structure

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

```
MADriverArgPacketStruct    STRUC
    ReturnAddr     DD  ?   ; [IN] Return address. Allows driver to retrun via RET.
    HeaderPtr      DD  ?   ; [IN] Pointer to the BIOS Driver Header (Reference 8.2.1.2).
    FunctionNum    DB  ?   ; [IN] Function number identifing the function to perform.
MADriverArgPacketStruct    ENDS
```

8.2.2.3 Parameters and Structures

8.2.2.3.1 Parameter pbInBuf

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

8.2.2.3.2 Parameter dwInPCRLen

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

8.2.2.3.3 Parameter bMAInitTPMFctId

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

8.2.2.3.4 Parameter bMAPhyPresenceTPMCmdId

BYTE *bMAPhyPresenceTPMCmdId*	
Description	Selects the TPM-Operation for the PhysicalPresence command. This value is used in the TPM-Param-Block of the TPM_PhysicalPresence command. For the detailed definition of this identifier please use the TCPA Main Specification.

8.2.2.4 MA Driver Functions

8.2.2.5 MA Driver Function Interface

The function number is contained in the FunctionNum field of the MADriverArgPacketStruct structure (Reference Section 8.2.2.2). The base for the function numbers is **01h**. The offset for vendor specific driver function numbers is 80h. All functions return their exit code in the DL Register.

8.2.2.5.1 Function MAInitTPM (Function Number: 01h)

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

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

BYTE MAInitTPM (**BYTE** *bMAInitTPMFctId*);	
Input Parameters	*DL = bMAInitTPMFctId* Function identifier for the TPM_Startup operation (see 8.2.2.3.3).
Return Value	*DL = return value of this function* One of the following values: **TPM_OK** **TPM_IS_LOCKED** **TPM_NO_RESPONSE** **TPM_INVALID_RESPONSE** **TPM_RESPONSE_TIMEOUT** **TPM_INVALID_ACCESS_REQUEST** **TPM_FIRMWARE_ERROR** **TPM_GENERAL_ERROR** **TPM_TRANSFER_ABORT** **TPM_TCG_COMMAND_ERROR**

8.2.2.5.2 Function MAHashAllExtendTPM (Function Number: 02h)

Start of informative comment:

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

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

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

End of informative comment.

BYTE MAHashAllExtendTPM (**DWORD** **pbInBuf*, **DWORD** *dwInPCRLen)*;	
Input Parameters	*EDX =*pbInBuf* Pointer to the start address of input buffer containing the data for the TPM device (see 8.2.2.3.1). *ECX = dwInPCRLen* PCRIndex and Length of the input buffer data (see 8.2.2.3.2).
Return Value	*DL = return value of this function* One of the following values: **TPM_OK** **TPM_IS_LOCKED** **TPM_NO_RESPONSE** **TPM_INVALID_RESPONSE** **TPM_RESPONSE_TIMEOUT** **TPM_INVALID_ACCESS_REQUEST** **TPM_FIRMWARE_ERROR** **TPM_GENERAL_ERROR** **TPM_TRANSFER_ABORT** **TPM_TCG_COMMAND_ERROR**

8.2.2.5.3 Function MAPhysicalPresenceTPM (Function Number: 03h)

Start of informative comment:

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

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

End of informative comment.

BYTE MAPhysicalPresenceTPM (**BYTE** *bMAPhyPresenceTPMCmdId*);	
Input Parameters	*DL = bMAPhyPresenceTPMCmdId* Command identifier for the TPM_PhysicalPresence operation (see 8.2.2.3.4).
Return Value	*DL = return value of this function* One of the following values: **TPM_OK** **TPM_IS_LOCKED** **TPM_NO_RESPONSE** **TPM_INVALID_RESPONSE** **TPM_RESPONSE_TIMEOUT** **TPM_INVALID_ACCESS_REQUEST** **TPM_FIRMWARE_ERROR** **TPM_GENERAL_ERROR** **TPM_TRANSFER_ABORT** **TPM_TCG_COMMAND_ERROR**

8.2.3 Memory Present (MP) Driver

8.2.3.1 Architecture

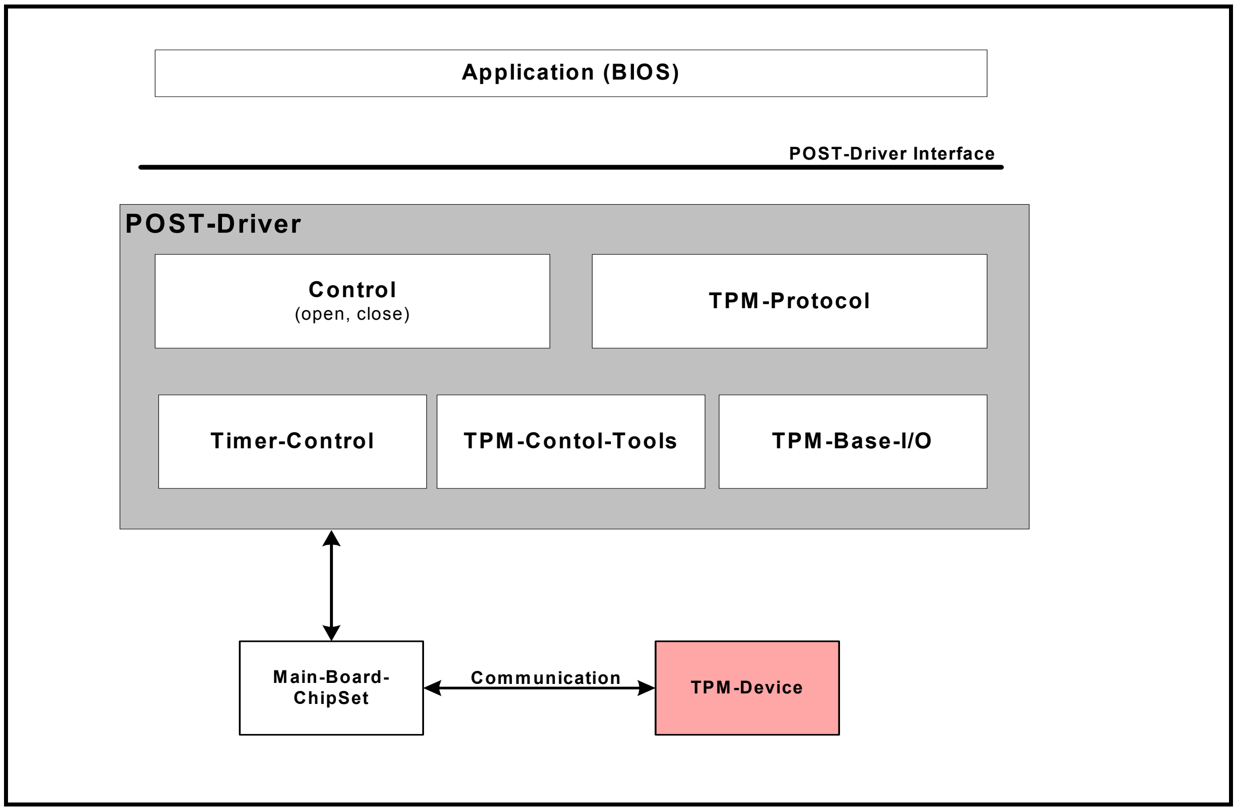
Start of informative comment:

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

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

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

End of informative comment.



■ Figure – 7.1 Pre-Boot Driver Interface

8.2.3.2 MP Driver Limitations

- No Interrupts are allowed. The MP driver MUST poll the TPM.
- The MP driver MAY be relocated after MAInitTPM and at any time between call MP driver functions.
- MP Driver needs to be placed into ACPI non-reclaimable area. The driver MUST support being relocated between calls.
- The resources allocated to the TPM MAY be changed by the BIOS between calling MP driver functions, therefore, the MAInitTPM function MUST be recallable.
- All registers not used for return parameters MUST be preserved.
- MP Driver needs to be built such that it has any data memory it requires is part of the body of the driver image.

8.2.3.3 Parameters and Structures

8.2.3.3.1 Parameter pbInBuf

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

8.2.3.3.2 Parameter pbOutBuf

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

8.2.3.3.3 Parameter dwInLen

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

8.2.3.3.4 Parameter dwOutLen

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

8.2.3.3.5 Structure TPMTransmitEntry

Start of informative comment:

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

End of informative comment.

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

The parameter pdwOutLen is both an input and output parameter:

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

8.2.3.3.6 Parameter lpTPMTransInfo

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

8.2.3.4 MP Driver function interface

The AL-Register contains the function selector number for the different functions of this driver (the base for this is **01h)**. The offset for vendor specific driver function numbers is 80h. All these functions returns there exit code in AL-Register.

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

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

This function does the initialization of the TPM and the driver and establishes (opens a connection) and verifies the communication (with the parameters from the header) between the POST-Driver and the TPM. If the interrupt number is set to FFh no interrupts are generated. This means the interrupts are disabled in the TPM device and the communication runs in polling mode this is the default mode.

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

BYTE MPInitTPM (void*)*;	
Input Parameters	*All necessary inputs are located in the driver header structure* (see 8.2.1.2).
Output Parameters	*None*
Return Value	*AL = return value of this function* One of the following values: **TPM_OK** **TPM_INVALID_ADR_REQUEST** **TPM_IS_LOOKED** **TPM_INVALID_DEVICE_ID** **TPM_INVALID_VENDOR_ID** **TPM_RESERVED_REG_INVALID** **TPM_FIRMWARE_ERROR** **TPM_UNABLE_TO_OPEN** **TPM_GENERAL_ERROR**

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

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

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

BYTE MPCloseTPM (void*)*;	
Input Parameters	*All necessary inputs are located in the driver header structure* (see 8.2.1.2).
Output Parameters	*None*
Return Value	*AL = return value of this function* One of the following values: **TPM_OK** **TPM_INVALID_ADR_REQUEST** **TPM_UNABLE_TO_CLOSE** **TPM_GENERAL_ERROR**

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

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

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

DWORD MPGetTPMStatusInfo (void*)*;	
Input Parameters	*All necessary inputs are located in the driver header structure* (see 8.2.1.2).
Output Parameters	*None*
Return Value	*EAX = return value of this function* For the coding of the return value see 8.2.3.5.

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

Transmits the data from the input buffer (**pbInBuf*) to the TPM and reads the response from the TPM to the output buffer (**pbOutBuf*). After successful Power-On and opening a TPM connection, the host can send the first request to the TPM by writing the bytes to the TPM. When the request is processed by the TPM and the response is available the TPM firmware issues an interrupt (or polling by the host if the interrupt is disabled) and the host can read it.

This function is responsible for block chaining and error handling during the interaction with the TPM device over communication interface.

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

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

BYTE MPTPMTransmit (**MPTPMTransmitEntryStruct** *lpTPMTransInfo*);*	
Input Parameters	*ESI = pointer to a* TPMTransmitEntryStruct (see 8.2.3.3.5). *pbInBuf* Pointer to the input buffer containing the data (TCG command string) for the TPM device (see 8.2.3.3.1). *dwInLen* Length of the input buffer data (see 8.2.3.3.3).
Input/Output Parameters	*pdwOutLen* Pointer to store the length info of the received data (see 8.2.3.3.4). It also carries the size (input) of the OutBuf to store the response of the TPM device.
Output Parameters	*pbOutBuf* Pointer to the output buffer to store the data from the TPM device (see 8.2.3.3.2).
Return Value	*AL = return value of this function* One of the following values: **TPM_OK** **TPM_IS_LOCKED** **TPM_NO_RESPONSE** **TPM_INVALID_RESPONSE** **TPM_RESPONSE_TIMEOUT** **TPM_INVALID_ACCESS_REQUEST** **TPM_FIRMWARE_ERROR** **TPM_GENERAL_ERROR** **TPM_TRANSFER_ABORT**

8.2.3.5 Return-Values for MPGetTPMStatusInfo (Function: 03h)

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

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

8.2.3.6 Error and Return Codes

The base number for the return codes is **TPM_RET_BASE = 01h.** The catalog of error and return codes can be extended to include TPM vendor specific return codes at the end of this list.

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

- Permanently disable the connection to the TPM,
- Take action to prevent the platform from loading the Operating System,
- Perform a Platform Reset, or
- Force transfer control of the platform to a manufacturer approved environment.

Error Code	Value	Description
TPM_OK	00h	Indicator of successful execution of the function.
TPM_GENERAL_ERROR	TPM_RET_BASE + 00	A general unidentified error occurred.
TPM_IS_LOCKED	TPM_RET_BASE + 01	The access cannot be granted the device is open.
TPM_NO_RESPONSE	TPM_RET_BASE + 02	No response from the TPM device.
TPM_INVALID_RESPONSE	TPM_RET_BASE + 03	The response from the TPM was invalid.
TPM_INVALID_ACCESS_REQUEST	TPM_RET_BASE + 04	The access parameters for this function are invalid.
TPM_FIRMWARE_ERROR	TPM_RET_BASE + 05	Firmware error during start up.
TPM_INTEGRITY_CHECK_FAILED	TPM_RET_BASE + 06	Integrity checks of TPM parameter failed.
TPM_INVALID_DEVICE_ID	TPM_RET_BASE + 07	The device ID for the TPM is invalid.
TPM_INVALID_VENDOR_ID	TPM_RET_BASE + 08	The vendor ID for the TPM is invalid.
TPM_UNABLE_TO_OPEN	TPM_RET_BASE + 09	Unable to open a connection to the TPM device.
TPM_UNABLE_TO_CLOSE	TPM_RET_BASE + 10	Unable to close a connection to the TPM device.
TPM_RESPONSE_TIMEOUT	TPM_RET_BASE + 11	Time out for TPM response.
TPM_INVALID_COM_REQUEST	TPM_RET_BASE + 12	The parameters for the communication access are invalid.
TPM_INVALID_ADR_REQUEST	TPM_RET_BASE + 13	The address parameter for the access is invalid.
TPM_WRITE_BYTE_ERROR	TPM_RET_BASE + 14	Bytes write error on the interface.
TPM_READ_BYTE_ERROR	TPM_RET_BASE + 15	Bytes read error on the interface.
TPM_BLOCK_WRITE_TIMEOUT	TPM_RET_BASE + 16	Blocks write error on the interface.
TPM_CHAR_WRITE_TIMEOUT	TPM_RET_BASE + 17	Bytes write time out on the interface.
TPM_CHAR_READ_TIMEOUT	TPM_RET_BASE + 18	Bytes read time out on the interface.
TPM_BLOCK_READ_TIMEOUT	TPM_RET_BASE + 19	Blocks read error on the interface.
TPM_TRANSFER_ABORT	TPM_RET_BASE + 20	Transfer abort in communication with TPM device.
TPM_INVALID_DRV_FUNCTION	TPM_RET_BASE + 21	Function number (AL-Register) invalid for this driver.
TPM_OUTPUT_BUFFER_TO_SHORT	TPM_RET_BASE + 22	Output buffer for the TPM response to short.
TPM_FATAL_COM_ERROR	TPM_RET_BASE + 23	Fatal error in TPM communication.
TPM_INVALID_INPUT_PARA	TPM_RET_BASE + 24	Input parameter for the function invalid.
TPM_TCG_COMMAND_ERROR	TPM_RET_BASE + 25	Error during execution of a TCG command.
TPM_VENDOR_BASE_RET	**128**	Start point for return codes are reserved for use by TPM vendors.

8.3 Physical Presence

Start of informative comment:

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

End of informative comment.

The Motherboard MAY provide a mechanism that provides proof of a human's physical presence to the Platform.

8.3.1 Physical Switch

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

8.3.2 Indication of Physical Presence from the CRTM

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

- Physical Presence NOT indicated: Exit normally, processing the remaining portions of the pre-boot environment.

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

- Physical Presence IS indicated: Transfer control of the platform to the utility that requires physical presence.

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