TCG

EFI Protocol Specification

Family "2.0"

Level 00 Revision 00.09

July 17, 2015

Committee Draft

Contact: admin@trustedcomputinggroup.org

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

TCG Public Review

TCG

Disclaimers, Notices, and License Terms

Change History

Revision	Date	Description
1	2/14/2015	Initial skeleton from the previous released version of the EFI Protocol Specification
2	2/25/2015	• Incorporated text on events after GetEventLog is called • Added Bibliography and moved some references to bibliography
3	2/27/2015	• Format changes • Incorporated Feedback
4	3/1/2015	• Renamed protocol
5	3/3/2015	• Renamed protocol
6	5/1/2015	• Updated references • Incorporated feedback • Added operation to retrieve response of SetActivePcrBanks service
7	5/10/2015	• Feedback on GetResultOfSetActivePcrBanks (keep result beyond lifetime of protocol)
8	5/11/2015	• Feedback on GetResultOfSetActivePcrBanks • Added flow for getting log through API and event table • Updated structure and protocol version of GetCapability output (adding GetResultOfSetActivePcrs changed the protocol)
9	5/13/2015	• Address comments from Cisco (use hexadecimal offsets)

Contents

List of Figures

List of Tables

Corrections and Comments

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

1 Introduction and Concepts

The purpose of this document is to define a standard interface to the TPM on an EFI platform. This standard interface is useful on any instantiations of an EFI platform that conforms to the EFI Specification. This EFI Protocol Specification is a pure interface specification that provides no information on “how” to construct the underlying firmware implementation.

OS loaders and OS manageability agents will use this interface to measure and log the boot process on EFI platforms.

1.1 Interoperability

Although this specification is for TPM 2.0 devices, it contains nothing that actively prevents the use of the specified protocol with TPM 1.2 devices.

2 References

The following documents are referenced in this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including amendments) applies.

1. *TPM Library Specification; Family 2.0; Level 00; Revision 01.16 (http://www.trustedcomputinggroup.org/resources/tpm_library_specification)* or later
2. *TCG PC Client Specific Platform TPM Profile for TPM 2.0 Version 1.00, Revision 0.43 (http://www.trustedcomputinggroup.org/resources/pc_client_platform_tpm_profile_ptp_specification)* or later
3. *UEFI Specification version 2.4 (Errata B) (http://uefi.org/specifications)* or later.
4. *TCG PC Client Specific Platform Firmware Profile Specification Family 2.0*.
5. *TCG Physical Presence Interface Specification, Family 1.2 and 2.0, Version 1.30, revision 0.52 (http://www.trustedcomputinggroup.org/resources/tcg_physical_presence_interface_specification)* or later
6. *TCG Algorithm Registry, version 1.22 (http://www.trustedcomputinggroup.org/resources/tcg_algorithm_registry)* or later
7. *TCG Vendor ID Registry, version 1.0, Revision 0.7 (http://www.trustedcomputinggroup.org/resources/vendor_id_registry)* or later

This specification also mentions the Microsoft Corporation, "*Windows Authenticode Portable Executable Signature Format*," Version 1.0, March 21, 2008.

3 Conventions

For the purpose of this document the following conventions apply.

3.1 Data Structure Descriptions

All constants and data SHALL be represented as little-endian bit format, which requires the low-order bit on the far left of a constant or data item and the high-order bit on the far right. Exceptions to this, if any, will be explicit in this specification.

All strings SHALL be represented as an array of ASCII bytes with the left-most character placed in the lowest memory location. All strings SHALL be zero terminated unless the containing byte array has a size limitation.

In some memory layout descriptions, certain fields are marked reserved. Software must initialize such fields to zero, and ignore them when read. On an update operation, software must preserve any reserved field.

All structures defined in this specification are packed. Some compilers may insert space between fields of a structure to align them. This functionality SHALL be disabled, so structures are packed.

3.2 Typographic Conventions

This document uses the following typographic conventions to illustrate programming concepts:

`Prototype` This typeface indicates prototype code.

`Argument` This typeface indicates arguments.

4 Abbreviations and Terminology

This specification uses the following abbreviations and terms:

Boot Services

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) The collection of interfaces and protocols that are present in the boot environment. The services minimally provide an OS loader with access to platform capabilities required to complete OS boot. Services are also available to drivers and applications that need access to platform capability. Boot services are terminated once the operating system takes control of the platform.

Boot Services Time

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) The period of time between platform initialization and the call to ExitBootServices(). During this time, EFI Drivers and applications are loaded iteratively and the system boots from an ordered list of EFI OS loaders.

CHAR16

The common EFI data type that is a 2-byte character. Unless otherwise specified, all strings are stored in the UTF-16 encoding format, as defined by Unicode 2.1 and ISO/IEC 10646 standards. Note: This definition is from Table 2-2 of the Extensible Firmware Specification, version 1.10, December 1, 2002.

EFI Driver

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) A module of code typically inserted into the firmware via protocol interfaces. Drivers may provide device support during the boot process or they may provide platform services. It is important not to confuse drivers in this specification with OS drivers that load to provide device support once the OS takes control of the platform.

EFI Hard Disk

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) A hard disk that supports the new EFI partitioning scheme.

EFI OS Loader

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) The first piece of operating system code loaded by the firmware to initiate the OS boot process. This code is loaded at a fixed address and then executed. The OS takes control of the system prior to completing the OS boot process by calling the interface that terminates all boot services.

Event Services

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) The set of functions used to manage EFI events. Includes **CheckEvent ( )**, **CreateEvent ( )**, **CloseEvent ( )**, **SignalEvent ( )**, and **WaitForEvent ( )**.

GPT

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) GUID'd Partition Table: A data structure that describes one or more partitions. It consists of a GPTHeader and, typically, at least one GPTPartition Entry. There are two GUID partition tables: the Primary Partition Table (located in LBA 1 of the disk) and a Backup Partition Table (located in the last LBA of the disk). The Backup Partition Table is a copy of the Primary Partition Table.

GUID

Globally Unique Identifier: A 128-bit value used to differentiate services and structures in the boot services environment.

Image

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) Either: (1) An executable file stored in a file system that complies with this specification. Images may be drivers, applications or OS loaders. Also called an EFI Image. (2) Executable binary file containing EBC and data. Output by the EBC linker.

Image Handle

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) A handle for loading an image; image handles support the loaded image protocol

Image Handoff State

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) The information handed off to a loaded image as it begins execution; it consists of the image's handle and a pointer to the image's system table.

Protocol

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) The information that defines how to access a certain type of device during boot services. A protocol consists of a Globally Unique Identifier (GUID), a protocol revision number, and a protocol interface structure. The interface structure contains data definitions and a set of functions for accessing the device.

A device can have multiple protocols. Each protocol is accessible through the device's handle.

System Table

(This definition is copied and pasted from the EFI 2.4 Specification, for the convenience of the reader) Table that contains the standard input and output handles for a UEFI application, as well as pointers to the boot services and runtime services tables. It may also contain pointers to other standard tables such as the ACPI, SMBIOS, and SAL System tables. A loaded image receives a pointer to its system table when it begins execution. Also called the EFI System Table.

TPM

Trusted Platform Module

UINT8, UINT16, UINT32

Basic types used in this specification to describe unsigned integers of various sizes. The number at the end of the type defines the size of the type in bits.

Variable

Unicode / GUID pair that is used to index persistent store in EFI

5 Event Log Structure

This section describes the layout of an event contained in the event log that is returned by the GetEventLog operation (Section 6.5). This section is for informative purposes only to provide the reader with all required information in one document. The normative description of the event layout can be found in the TCG PC Client EFI Platform Specification.

Previous specifications describing the format of the boot event log, mandated the use of SHA1 to calculate digests of events. This document refers to this event log format as SHA1 log format. This section defines the layout for a crypto agile log format that allows the use of hashing algorithms in addition to SHA1.

The crypto agile event log entry format introduced below uses a variable sized field for the list of digests. To allow a parser to parse the log format, even if it contains digests for algorithms unknown to the parser, the size of all used digests has to be defined. To avoid a recursive problem, the log header is defined to be in SHA1 event log entry format.

5.1 SHA1 Event Log Entry Format

An event log entry (or just event) is defined to be of the following format:

```
typedef struct tdTCG_PCR_EVENT {
  TCG_PCRINDEX   PCRIndex;       //PCRIndex event extended to
  TCG_EVENTTYPE  EventType;      //Type of event (see EFI specs)
  TCG_DIGEST     Digest;         //Value extended into PCRIndex
  UINT32         EventSize;      //Size of the event data
  UINT8          Event[EventSize]; //The event data
} TCG_PCR_EVENT;                 //Structure to be added to the
                                 //Event Log

typedef UINT32 TCG_PCRINDEX;

typedef UINT32 TCG_EVENTTYPE;

typedef UINT8 TCG_DIGEST[20];
```

The SHA1 digest of the event data (Event field in TCG_PCR_EVENT) or of external data is stored in the Digest field of the TCG_PCR_EVENT and extended into the SHA1 PCR identified by the PCRIndex field of the TCG_PCR_EVENT structure.

5.2 Crypto Agile Log Entry Format

To accommodate TPM devices with PCR banks that use other hashing algorithms than SHA1, the event log entry format has been changed. It replaces the fixed sized Digest field with a list of tagged digests.

```
typedef struct tdTCG_PCR_EVENT2 {
  TCG_PCRINDEX        PCRIndex;       //PCRIndex event extended to
  TCG_EVENTTYPE       EventType;      //Type of event (see [2])
  TPML_DIGEST_VALUES  Digests;        //List of digests extended to
                                      //PCRIndex
  UINT32              EventSize;      //Size of the event data
  UINT8               Event[EventSize];  //The event data
} TCG_PCR_EVENT2;                     //Structure to be added to the
                                      //Event Log

typedef UINT32 TCG_PCRINDEX;

typedef UINT32 TCG_EVENTTYPE;



typedef struct tdTPML_DIGEST_VALUES {
    UINT32      Count;         // number of digests
    TPMT_HA     Digests[Count]; // Count digests
} TPML_DIGEST_VALUES;

typedef struct tdTPMT_HA {
    UINT16      AlgorithmId;  // ID of hashing algorithm
    UINT8       Digest[];     // Digest, depends on AlgorithmId
} TPMT_HA;
```

Note that although the type names from the TPM 2.0 Library Specification are used, the encoding of the count member and the AlgorithmID are little-endian, as is the rest of the log format.

An event will be densely packed, that is, even though there can be multiple digests in an event, the algorithm ID of the next digest follows immediately after the last byte of the previous digest. See the second example below for an illustration of the offsets in the event log.

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

Table 1: Example 1 of Crypto Agile Log Event

Field Name	Offset	Size (in bytes)	Content
PCRIndex	0x00	4	2
EventType	0x04	4	EV_SEPARATOR (4)
Digests	0x08		

Field Name	Offset	Size (in bytes)	Content
Digests.Count	0x08	4	1
Digests.Digests	0x0C		
Digests.Digests[0].AlgorithmID	0x0C	2	SHA1 (4)
Digests.Digests[0].Digest	0x0E	20	0x90, 0x69 0xca, 0x78, 0xe7, 0x45, 0x0a, 0x28, 0x51, 0x73, 0x43, 0x1b, 0x3e, 0x52, 0xc5, 0xc2, 0x52, 0x99, 0xe4, 0x73
EventSize	0x22	4	4
Event	0x26	4	0x00, 0x00, 0x00, 0x00

The encoding as byte stream would look like follows. The start of the line describes the offset for the first byte in the line.

```
0000: 02 00 00 00 04 00 00 00 – 01 00 00 00 04 00 90 69
0010: ca 78 e7 45 0a 28 51 73 – 43 1b 3e 52 c5 c2 52 99
0020: e4 73 04 00 00 00 00 00 – 00 00
```

The following is the same separator event using two PCR banks:

Table 2: Example 2 of Crypto Agile Log Event

Field Name	Offset	Size (in bytes)	Content
PCRIndex	0x00	4	2
EventType	0x04	4	EV_SEPARATOR (4)
Digests	0x08		
Digests.Count	0x08	4	2
Digests.Digests	0x0C		
Digests.Digests[0].AlgorithmID	0x0C	2	SHA1 (4)
Digests.Digests[0].Digest	0x0E	20	0x90, 0x69 0xca, 0x78, 0xe7, 0x45, 0x0a, 0x28, 0x51, 0x73, 0x43, 0x1b, 0x3e, 0x52, 0xc5, 0xc2, 0x52, 0x99, 0xe4, 0x73
Digests.Digests[1].AlgorithmID	0x22	2	SHA-256 (0xb)

Field Name	Offset	Size (in bytes)	Content
Digests.Digests[1].Digest	0x24	32	0xdf, 0x3f, 0x61, 0x98, 0x04, 0xa9, 0x2f, 0xdb, 0x40, 0x57, 0x19, 0x2d, 0xc4, 0x3d, 0xd7, 0x48, 0xea, 0x77, 0x8a, 0xdc, 0x52, 0xbc, 0x49, 0x8c, 0xe8, 0x05, 0x24, 0xc0, 0x14, 0xb8, 0x11, 0x19
EventSize	0x44	4	4
Event	0x48	4	0x00, 0x00, 0x00, 0x00

The second example as byte stream looks like this:

```
0000: 02 00 00 00 04 00 00 00 – 02 00 00 00 04 00 90 69
0010: ca 78 e7 45 0a 28 51 73 – 43 1b 3e 52 c5 c2 52 99
0020: e4 73 0B 00 df 3f 61 98 – 04 a9 2f db 40 57 19 2d
0030: c4 3d d7 48 ea 77 8a dc – 52 bc 49 8c e8 05 24 c0
0040: 14 b8 11 19 04 00 00 00 – 00 00 00 00
```

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

5.3 Event Log Header

To allow parsers to identify the log format based on the content of the log, the first event of the log is formatted as a TCG_PCR_EVENT structure independent of the format for the rest of the log. A parser may read the first event of type TCG_PCR_EVENT and because of its fixed size, easily find the event data. The fields of the event log header are defined to be PCRIndex of 0, EventType of EV_NO_ACTION, Digest of 20 bytes of 0, and Event content defined as TCG_EfiSpecIDEventStruct. This first event is the event log header.

The TCG PC Client Specific Platform Firmware Profile contains a definition for the content of the event log header – the TCG_EfiSpecIDEventStruct structure. This document contains a copy for reference:

```
typedef struct tdTCG_EfiSpecIdEventStruct {
  BYTE[16]              signature;
  UINT32                platformClass;
  UINT8                 specVersionMinor;
  UINT8                 specVersionMajor;
  UINT8                 specErrata;
  UINT8                 uintnSize;
  UINT32                numberOfAlgorithms;
  TCG_EfiSpecIdEventAlgorithmSize[numberOfAlgorithms] digestSizes;
  UINT8                 vendorInfoSize;
  BYTE[VendorInfoSize]  vendorInfo;
} TCG_EfiSpecIDEventStruct;
```

Where the type TCG_EfiSpecIdEventAlgorithmSize structure is defined as:

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

The specification version field in the TCG_EfiSpecIDEventStruct defines which field of the structure are valid. The minimum version for the above version of the structure are specVersionMajor = 2, specVersionMinor = 0, specErrata = 0.

Table 3: Description of fields in spec ID event

Type	Name	Description
BYTE[16]	Signature	The null terminated ASCII string “Spec ID Event03”. SHALL be set to {0x53, 0x70, 0x65, 0x63, 0x20, 0x49, 0x44, 0x20, 0x45, 0x76, 0x65, 0x6e, 0x74, 0x30, 0x33, 0x00}.
UINT32	platformClass	The value for the Platform Class. The enumeration is

Type	Name	Description
		defined in the TCG ACPI Specification Client Common Header.
UINT8	specVersionMinor	The TCG EFI Platform Specification minor version number this BIOS supports. Any BIOS supporting this version (2.0) SHALL set this value to 0x00.
UINT8	specVersionMajor	The TCG EFI Platform Specification major version number this BIOS supports. Any BIOS supporting this version (2.0) SHALL set this value to 0x02.
UINT8	specErrata	The TCG EFI Platform Specification errata for this specification this BIOS supports. Any BIOS supporting this version and errata (2.0) SHALL set this value to 0x00.
UINT8	uintnSize	Specifies the size of the UINTN fields used in various data structures used in this specification. 0x01 indicates UINT32 and 0x02 indicates UINT64.
UINT32	numberOfAlgorithms	The number of hashing algorithms used in this event log (except the first event). All events in this event log use all hashing algorithms defined here.
EfiSpecIdEventAlgorithmSize[]	digestSizes	An array of size numberOfAlgorithms of value pairs. Each value pair consists of two UINT16 members. The first member is a TCG defined hashing algorithm ID. The second member is the size of the digest for the respective hashing algorithm. If the log

Type	Name	Description
		contains SHA1 digest, one value pair would be { 0x4, 0x14 }.
UINT8	vendorInfoSize	Size in bytes of the VendorInfo field. Maximum value SHALL be FFh bytes.
BYTE[]	vendorInfo	Provided for use by the BIOS implementer. The value might be used, for example, to provide more detailed information about the specific BIOS such as BIOS revision numbers, etc. The values within this field are not standardized and are implementer-specific. Platform-specific or -unique information SHALL NOT be provided in this field.

A log parser can use the information in digestSizes to iterate the TCG_PCR_EVENT2 elements in the remainder of the crypto agile log. A parser performs the following steps for each TCG_PCR_EVENT2:

1. Read 4 bytes as PCRIndex.
2. Read 4 bytes as EventType.
3. Read 4 bytes as count of digest values.
4. For each digest value:
    a. Read 2 bytes as algorithm ID of the current digest value
    b. Look up the digest size for this algorithm ID using the digestSizes table from the log header.
    c. Read the number of bytes determined in the previous step as digest.
5. Read 4 bytes as EventSize.
6. Read the number of bytes determines in the previous step as EventData.

All crypto agile events have the digests listed in the same sequence. That is, if the first crypto agile log has the SHA1 digest appear first followed by the SHA-256 digest, all subsequent events also have the SHA1 digest appear first followed by the SHA-256 digest.

When the GetEventLogs operation (Section 6.5) is called, all events must contain digests for all active PCR banks, i.e. hashing algorithms. That is, if the log header event defines digest sizes for SHA1 and SHA256 hashing algorithms, all events must contain SHA1 and SHA256 digests.

5.4 PCR Banks

A Platform Configuration Register (PCR) is a memory location in the TPM that has some unique properties. The size of the value that can be stored in a PCR is determined by the size of a digest generated by an associated hashing algorithm. A SHA-1 PCR would be a PCR that can store 20 bytes – the size of a SHA-1 digest. To store a new value in a PCR, the existing value is extended with a new value as follows: The existing value is concatenated with the argument of the Extend operation. The resulting concatenated value is then used as input to the associated hashing algorithm, which generates a digest of the concatenated value. This computed digest becomes the new value of the PCR.

$$PCR_n = HASH_{alg}(PCR_{n-1} \,||\, ArgumentOfExtend)$$

The argument to the extend operation is of the size of the digest of the hashing algorithm associated with the PCR.

Multiple PCR that are associated with the same hashing algorithm are usually referred to as a PCR bank.

The *TCG PC Client Specific Platform TPM Profile for TPM 2.0* defines that there should be at least one PCR bank with 24 registers, of which the first 16 can only be reset to a well-defined initial value by resetting the TPM. This way the TPM can ensure that the value of a PCR can only be modified via the Extend operation.

Each of the PCR within a PCR bank is addressed through its index – the PCR index. Each PCR at an index contains the sequence of extended digests of a subgroup of events. The *TCG PC Client Specific Platform Firmware Profile Specification Family 2.0* defines the measurements for each PCR index. Usually an array notation is used to differentiate the PCR in a bank: PCR at index zero is written as PCR[0]. Each PCR bank has PCR for each index. If the SHA-1 and SHA-256 PCR banks are active, there is a PCR[0] for SHA-1 and a PCR[0] for SHA-256.

The TPM PCR are used as checksums of all log events that are defined to be extended (or measured) in the TPM. A validator can compute the expected PCR values and compare them to the PCR values of the TPM. Because the TPM PCR cannot be modified arbitrarily, a match between expected PCR value and TPM PCR value indicates an unmodified log.

5.5 Notes to the implementer

The implementer of this interface can determine the values for the digest sizes as follows: Firmware uses the TPM 2.0 GetCapability command to identify the active PCR banks. It can then call the TPM2_Hash command with a zero value (for instance) for each algorithm. The returned value of TPM2_Hash allows the implementer to determine the size of the digest for this algorithm. Firmware can then use the algorithm ID and digest size values to generate the mapping for the log header event.

If the implementer uses the TPM2_Event command to hash and extend an event to all active PCR banks in one operation, the return value is a TPML_DIGEST_VALUES structure. This return value is formatted in TPM encoding and cannot be placed as is into the event log. The count field and the algorithmID field for the digests have to be re-encoded to little endian encoding. Because the implementer already determined the digest sizes for the algorithms above, it can parse the TPML_DIGEST_VALUES easily to re-encode them.

6 EFI TPM2 Protocol

The EFI TPM2 protocol is used to communicate with a TPM implementation in UEFI – to send commands to a TPM, use it for trusted operations, and to provide access to the firmware log of measurements extended in the TPM. The implementation of the protocol maintains an event log of measurements recorded in the TPM in SHA1 log format or crypto agile log format. Implementers may create additional event logs with other formats, but this version of the protocol does not define a way to retrieve them. Implementers may choose to store only one format and convert the log to the requested format.

UEFI provides mechanisms to retrieve the below protocol structure given the specified GUID. The function pointers in the protocol structure can then be used to invoke the various functions. This specifications defines the arguments to these functions, the functionality that should be implemented, and the return value.

The EFI TCG2 protocol SHALL use the following GUID.

GUID –

```
#define EFI_TCG2_PROTOCOL_GUID \
   {0x607f766c, 0x7455, 0x42be, 0x93, \
      0x0b, 0xe4, 0xd7, 0x6d, 0xb2, 0x72, 0x0f}
```

6.1 Protocol Version

A user of this protocol should call the `EFI_TCG2_PROTOCOL`.GetCapabilities operation (Section 6.4) to determine the functionality implemented by this interface. There are earlier implementations of this protocol that implement a subset of the functions and capabilities defined here.

6.2 Protocol Interface Structure

```
typedef struct tdEFI_TCG2_PROTOCOL {
  EFI_TCG2_GET_CAPABILITY        GetCapability;
  EFI_TCG2_GET_EVENT_LOG         GetEventLog;
  EFI_TCG2_HASH_LOG_EXTEND_EVENT HashLogExtendEvent;
  EFI_TCG2_SUBMIT_COMMAND        SubmitCommand;
  EFI_TCG2_GET_ACTIVE_PCR_BANKS  GetActivePcrBanks;
  EFI_TCG2_SET_ACTIVE_PCR_BANKS  SetActivePcrBanks;
  EFI_TCG2_GET_RESULT_OF_SET_ACTIVE_PCR_BANKS
                                 GetResultOfSetActivePcrBanks;
} EFI_TCG2_PROTOCOL;
```

Table 4: Protocol Interface Structure

Parameter	Description
GetCapability	This service provides information about the TPM and firmware capabilities
GetEventLog	Get a pointer to a firmware event log
HashLogExtendEvent	This service will cause the EFI TCG2 protocol driver to extend an event and (optionally) write the event to the crypto agile log.
SubmitCommand	This service submits a TPM command directly to the TPM.
GetActivePcrBanks	Returns a bitmap of currently active TPM PCR banks. (Only implemented for ProtocolVersion larger than or equal to 1.1.)
SetActivePcrBanks	Tries to set the active TPM PCR banks according to the provided bitmap. (Only implemented for ProtocolVersion larger than or equal to 1.1.)

6.3 Description

The EFI_TCG2_PROTOCOL abstracts TPM activity. This protocol instance provides a Boot Service and is instantiated as a Boot Service Driver.

Boot Service Drivers are terminated when ExitBootServices( ) is called and all memory resources consumed by the Boot Services Drivers are released for use in the operating system environment.

This Boot Service must create an EVT_SIGNAL_EXIT_BOOT_SERVICES event. This event will be notified by the system when ExitBootServices( ) is invoked.

EVT_SIGNAL_EXIT_BOOT_SERVICES is a synchronous event used to ensure that certain activities occur following a call to a specific interface function; in this case, that is the cleanup that needs to be done in response to the ExitBootServices( ) function. ExitBootServices( ) will not clean up after drivers that have been loaded. Drivers must clean up after themselves by creating an event with type EVT_SIGNAL_EXIT_BOOT_SERVICES and a Notification Function that is within the driver itself. Then, when ExitBootServices( ) has finished its cleanup, it signals the event type EVT_SIGNAL_EXIT_BOOT_SERVICES.

For implementation details about how a Boot Service instantiated as an EFI Driver creates this required EVT_SIGNAL_EXIT_BOOT_SERVICES event, see Section 6.1 of UEFI Specification 2.4.

6.4 EFI_TCG2_PROTOCOL.GetCapability

The EFI_TCG2_PROTOCOL GetCapability function call provides protocol capability information and state information.

6.4.1 Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG2_GET_CAPABILITY) (
  IN EFI_TCG2_PROTOCOL                *This,
  IN OUT EFI_TCG2_BOOT_SERVICE_CAPABILITY   *ProtocolCapability,
);
```

6.4.2 Parameters:

Table 5: GetCapability Parameters

Parameter	Description
This	Indicates the calling context.
ProtocolCapability	The caller allocates memory for a EFI_TCG2_BOOT_SERVICE_CAPABILITY structure and sets the size field to the size of the structure allocated. The callee fills in the fields with the EFI protocol capability information and the current EFI TCG2 protocol state information up to the number of fields which fit within the size of the structure passed in.

6.4.3 Related Definitions

The protocol capability structure allows a caller to determine which of the functions in the protocol can be called and which arguments can be used.

The event log format definitions specify if firmware supports the SHA-1 log format (EFI_TCG2_EVENT_LOG_FORMAT_TCG_1_2) or the crypto agile log format (EFI_TCG2_EVENT_LOG_FORMAT_TCG_2) or both (both bits set in a bitmask).

```
#define EFI_TCG2_EVENT_LOG_FORMAT_TCG_1_2 0x00000001

#define EFI_TCG2_EVENT_LOG_FORMAT_TCG_2   0x00000002


typedef UINT64 EFI_PHYSICAL_ADDRESS;

typedef UINT32 EFI_TCG2_EVENT_LOG_BITMAP;

typedef UINT32 EFI_TCG2_EVENT_LOG_FORMAT;

typedef UINT32 EFI_TCG2_EVENT_ALGORITHM_BITMAP;
```

```
typedef struct tdEFI_TCG2_VERSION {
   UINT8 Major;
   UINT8 Minor;
} EFI_TCG2_VERSION;

typedef struct tdEFI_TCG2_BOOT_SERVICE_CAPABILITY {
   UINT8                             Size;
   EFI_TCG2_VERSION                  StructureVersion;
   EFI_TCG2_VERSION                  ProtocolVersion;
   EFI_TCG2_EVENT_ALGORITHM_BITMAP   HashAlgorithmBitmap;
   EFI_TCG2_EVENT_LOG_BITMAP         SupportedEventLogs;
   BOOLEAN                           TPMPresentFlag;
   UINT16                            MaxCommandSize;
   UINT16                            MaxResponseSize;
   UINT32                            ManufacturerID;
   UINT32                            NumberOfPcrBanks;
   EFI_TCG2_EVENT_ALGORITHM_BITMAP   ActivePcrBanks;
} EFI_TCG2_BOOT_SERVICE_CAPABILITY;
```

The supported event log formats in the variable SupportedEventLogs are specified as bitmap using the values EFI_TCG2_EVENT_LOG_FORMAT_TCG_1_2 and EFI_TCG2_EVENT_LOG_FORMAT_TCG_2 as currently specified bits.

The hashing algorithms in the variables HashAlgorithmBitmap and ActivePcrBanks are specified as bitmaps. The hashing algorithm bitmaps are defined in the TCG Algorithm Registry and defined as follows:

```
#define EFI_TCG2_BOOT_HASH_ALG_SHA1      0x00000001

#define EFI_TCG2_BOOT_HASH_ALG_SHA256    0x00000002

#define EFI_TCG2_BOOT_HASH_ALG_SHA384    0x00000004

#define EFI_TCG2_BOOT_HASH_ALG_SHA512    0x00000008

#define EFI_TCG2_BOOT_HASH_ALG_SM3_256   0x00000010
```

Table 6: Boot Service Capability Fields

Field	Description
Size	Allocated size of the structure

Field	Description
StructureVersion	Version of the EFI_TCG2_BOOT_SERVICE_CAPABILITY structure itself. For this version of the protocol, the Major version SHALL be set to 1 and the Minor version SHALL be set to 1.
ProtocolVersion	Version of the EFI TCG2 protocol. For this version of the protocol, the Major version SHALL be set to 1 and the Minor version SHALL be set to 1.
HashAlgorithmBitMap	Supported hash algorithms (this bitmap is determined by the supported PCR banks in the TPM and the hashing algorithms supported by the firmware)
SupportedEventLogs	Bitmap of supported event log formats (see above)
TPMPresentFlag	False = TPM not present
MaxCommandSize	Max size (in bytes) of a command that can be sent to the TPM
MaxResponseSize	Max size (in bytes) of a response that can be provided by the TPM
ManufacturerID	4-byte Vendor ID (see *TCG Vendor ID registry*, Section “TPM Capabilities Vendor ID”)
NumberOfPcrBanks	Maximum number of PCR banks (hashing algorithms) supported. No granularity is provided to support a specific set of algorithms. Minimum value is 1.
ActivePcrBanks	A bitmap of currently active PCR banks (hashing algorithms). This is a subset of the supported hashing algorithms reported in HashAlgorithmBitMap. NumberOfPcrBanks defines the number of bits that are set.

6.4.4 Description

The EFI_TCG2_PROTOCOL Get Capability function call provides EFI protocol version and capability information as well as state information about the EFI TCG2 protocol. The caller SHALL set the Size field of the EFI_TCG2_BOOT_SERVICE_CAPABILITY structure allocated. It is expected future versions of this function call may add additional fields to the structure. The Size value passed in by the caller will determine which fields the function will be able to populate. For example:

```
ProtocolCapability.Size = sizeof(EFI_TCG2_BOOT_SERVICE_CAPABILITY);
```

For this version of the specification:

1. If the This or the ProtocolCapability parameters are NULL, the functional call will return EFI_INVALID_PARAMETER.

2. If the input ProtocolCapability.Size < size of the EFI_TCG2_BOOT_SERVICE_CAPABILITY up to and including the vendor ID

field, the function will set ProtocolCapability.Size equal to size of the EFI_TCG2_BOOT_SERVICE_CAPABILITY up to and including the vendor ID field and will return the error code EFI_BUFFER_TOO_SMALL, the values of the remaining fields will be undefined.

3. If the input ProtocolCapability.Size < sizeof(EFI_TCG2_BOOT_SERVICE_CAPABILITY) the function will initialize the fields included in ProtocolCapability.Size The values of the remaining fields will be undefined.

4. The following return values SHALL be set:

```
ProtocolCapability.StructureVersion.Major = 1
ProtocolCapability.StructureVersion.Minor = 3
ProtocolCapability.ProtocolVersion.Major = 1
ProtocolCapability.ProtocolVersion.Minor = 3
```

5. If the platform does not have a TPM then the following values SHALL be returned:

```
ProtocolCapability.SupportedEventLogs = 0
ProtocolCapability.HashAlgorithmBitmap = 0
ProtocolCapability.TPMPresentFlag = FALSE
ProtocolCapability.MaxCommandSize = 0
ProtocolCapability.MaxResponseSize = 0
ProtocolCapability.ManufacturerID = 0
ProtocolCapability.NumberOfPcrBanks = 0
ProtocolCapability.ActivePcrBanks = 0
```

6.4.5 Status Codes Returned:

Table 7: GetCapability Return Values

Return Code	Description
EFI_SUCCESS	Operation completed successfully.
EFI_DEVICE_ERROR	The command was unsuccessful. The ProtocolCapability variable will not be populated.
EFI_INVALID_PARAMETER	One or more of the parameters are incorrect. The ProtocolCapability variable will not be populated.
EFI_BUFFER_TOO_SMALL	The ProtocolCapability variable is too small to hold the smallest sized response. The capability structure argument will be partially populated (required Size field will be set).

6.5 EFI_TCG2_PROTOCOL.GetEventLog

The EFI_TCG2_PROTOCOL Get Event Log function call allows a caller to retrieve the address of a given event log and its last entry.

6.5.1 Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG2_GET_EVENT_LOG) (
  IN  EFI_TCG2_PROTOCOL          *This,
  IN  EFI_TCG2_EVENT_LOG_FORMAT  EventLogFormat,
  OUT EFI_PHYSICAL_ADDRESS       *EventLogLocation,
  OUT EFI_PHYSICAL_ADDRESS       *EventLogLastEntry,
  OUT BOOLEAN                    *EventLogTruncated
);
```

6.5.2 Parameters

Table 8: GetEventLog Parameters

Parameter	Description
EventLogFormat	The type of the event log for which the information is requested.
EventLogLocation	A pointer to the memory address of the event log.
EventLogLastEntry	If the event log contains more than one entry, this is a pointer to the address of the start of the last entry in the event log in memory. For information about what values are returned in this parameter in the special cases of an empty event log or an event log with only one entry, see the Description section below.
EventLogTruncated	If the event log is missing at least one entry because an event would have exceeded the area allocated for events, this value is set to TRUE. Otherwise, the value will be FALSE and the event log will be complete.

6.5.3 Description

The firmware manages an event log of the measurements recorded in the TPM during the boot process. During the boot process, before UEFI platform initialization, an entry is made in the event log for each measurement extended in the TPM. In the UEFI environment, each time a call is made to HashLogExtendEvent to extend a measurement in the TPM, an event is generally recorded in the event log containing the extended measurement. If the area allocated by firmware for the event log was too small to hold

all events added, the function call indicates the event log was truncated and has missing entries.

The event log area returned by this function is released when ExitBootServices( ) is called. Callers of this method SHALL not access the area after ExitBootServices( ) has been called. For this version of the specification:

1. If EventLogFormat does not equal EFI_TCG2_EVENT_LOG_FORMAT_TCG_1_2 or EFI_TCG2_EVENT_LOG_FORMAT_TCG_2, the function call SHALL return EFI_INVALID_PARAMETER.
2. If the EventLogFormat does equal EFI_TCG2_EVENT_LOG_FORMAT_TCG_1_2 and one of the currently active PCR banks is the SHA1 bank, the function SHALL return a log conforming to the SHA1 log format. If EventLogFormat does equal EFI_TCG2_EVENT_LOG_FORMAT_TCG_1_2 and SHA1 is not an active PCR bank, the function SHALL return EFI_INVALID_PARAMETER.
3. If the EventLogFormat does equal EFI_TCG2_EVENT_LOG_FORMAT_TCG_2, the function SHALL return a log conforming to the crypto agile log format.
4. If no TPM is present the function SHALL set the following values and return EFI_SUCCESS:

   ```
   EventLogLocation = NULL

   EventLogLastEntry = NULL

   EventLogTruncated = FALSE
   ```

5. The EventLogLocation value SHALL be set to the start of the event log specified by the requested format in memory.
6. If the specified event log:
   1. does not contain any events then EventLogLastEntry SHALL be set to 0
   2. contains exactly one entry then EventLogLastEntry SHALL be set to the same value as EventLogLocation
   3. contains more than one event then EventLogLastEntry SHALL be set to the start address of the last event of the specified event log
7. If a prior call to EFI_TCG2_PROTOCOL.HashLogExtendEvent returned EFI_VOLUME_FULL then EventLogTruncated SHALL be set to TRUE, otherwise it SHALL be set to FALSE.

6.5.4 Status Codes Returned

Table 9: GetEventLog Return Values

Return Code	Description
EFI_SUCCESS	Operation completed successfully.
EFI_INVALID_PARAMETER	One or more of the parameters are incorrect (e.g. asking for an event log whose format is not supported).

6.6 EFI_TCG2_PROTOCOL.HashLogExtendEvent

The EFI_TCG2_PROTOCOL HashLogExtendEvent function call provides callers with an opportunity to extend and optionally log events without requiring knowledge of actual TPM commands. The extend operation will occur even if this function cannot create an event log entry (e.g. due to the event log being full).

6.6.1 Prototype

```
typedef
EFI_STATUS
(EFIAPI * EFI_TCG2_HASH_LOG_EXTEND_EVENT) (
  IN EFI_TCG2_PROTOCOL     *This,
  IN UINT64                Flags,
  IN EFI_PHYSICAL_ADDRESS  DataToHash,
  IN UINT64                DataToHashLen,
  IN EFI_TCG2_EVENT        *EfiTcgEvent,
);
```

6.6.2 Parameters

Table 10: HashLogExtendEvent Parameters

Parameter	Description
This	Indicates the calling context.
Flags	Bitmap providing additional information (see below).
DataToHash	Physical address of the start of the data buffer to be hashed.
DataToHashLen	The length in bytes of the buffer referenced by DataToHash.
EfiTcgEvent	Pointer to data buffer containing information about the event.

6.6.3 Related Definitions

The EFI_TCG2_EVENT_HEADER data structure contains information relevant when a digest should be extended – the PCRIndex. If firmware generates a log entry for this event the EventType entry and PCRIndex entry can be used to fill the header of the log event. The digest that is computed from the DataToHash is used to fill the Digests field in the log entry. The content of the Event field is then used to fill the EventData field of the log entry. The size of the Event field is determined by subtracting the value of the HeaderSize field and size of the Size field (sizeof(UINT32)) from the value of the Size field.


```
typedef struct tdEFI_TCG2_EVENT {
  UINT32                Size;
  EFI_TCG2_EVENT_HEADER Header;
  UINT8                 Event[];
} EFI_TCG2_EVENT;

typedef struct tdEFI_TCG2_EVENT_HEADER {
  UINT32                HeaderSize;
  UINT16                HeaderVersion;
  TCG_PCRINDEX          PCRIndex;
  TCG_EVENTTYPE         EventType;
} EFI_TCG2_EVENT_HEADER;

typedef UINT32 TCG_PCRINDEX;

typedef UINT32 TCG_EVENTTYPE;
```

Table 11: EFI TCG2 Event Field Descriptions

Field	Description
Size	Total size of the event including the Size component, the header and the Event data.
HeaderSize	Size of the event header itself (sizeof(EFI_TCG2_EVENT_HEADER)).
HeaderVersion	Header version. For this version of this specification, the value SHALL be 1.
PCRIndex	Index of the PCR that is extended (0 – 23).
EventType	Type of the event that is extended (and optionally logged).

6.6.4 Flag Values

The Flags variable is a bitmap providing additional data as follows:

```
#define EFI_TCG2_EXTEND_ONLY  0x0000000000000001
```

This bit SHALL be set when an event SHALL be extended but not logged.

```
#define PE_COFF_IMAGE         0x0000000000000010
```

This bit SHALL be set when the intent is to measure a PE/COFF image.

6.6.5 Description

The EFI_TCG2_PROTOCOL Hash Log Extend Event function call calculates the measurement of a data buffer (possibly containing a PE/COFF binary image) and causes

the EFI TCG2 protocol driver to extend the measurement. In addition, the service optionally creates an event log entry and appends it to the event log for each event log format supported by the service. The service allows a caller to make use of the TPM without knowing anything about specific TPM commands.

The use of this function to measure PE/COFF images must be done before relocations have been applied to the image. Note: Use caution using this method to measure PE/COFF images. Generally implementations that load PE/COFF images strip important data during the load process from the image and may change the image section alignment in memory. The net result is calculating the hash of an in-memory image does not match the actual measurement for the image as properly calculated when it is loaded from storage media.

Upon invocation, the function SHALL perform the following actions:

1. If any of the parameters This, DataToHash, or EfiTcgEvent are NULL, the function SHALL return EFI_INVALID_PARAMETER.
2. If the EfiTcgEvent.Size is less than EfiTcgEvent.Header.HeaderSize + sizeof(UINT32), the function SHALL return EFI_INVALID_PARAMETER.
3. If the EfiTcgEvent.Header.PCRIndex is not 0 through 23, inclusive, the function SHALL return EFI_INVALID_PARAMETER.
4. If the Flags bitmap has the PE_COFF_IMAGE bit SET but the PE/COFF image is corrupt or not understood the function SHALL return EFI_UNSUPPORTED.
5. The function allows any value for the EfiTcgEvent.Header.EventType parameter.
6. The function SHALL calculate the digest (measurement) of the data starting at DataToHash with a length of DataToHashLen. When measuring a PE/COFF image, the EventType SHALL be as defined in TCG PC Client Specific Platform Specification (for example, when measuring an EFI Boot Application, the EventType SHALL be EV_EFI_BOOT_SERVICES_APPLICATION) and the EfiTcgEvent value SHALL be the value of the EFI_IMAGE_LOAD_EVENT structure.

   The HashLogExtendEvent service SHALL hash the PE/COFF image in accordance with the procedure specified in “Calculating the PE Image Hash” section of the “*Windows Authenticode Portable Executable Signature Format*” document. Note that the function can use the TPM to calculate the digest using TPM2_PCR_Event.
7. The function SHALL successfully send the TPM2_PCR_Extend command to the TPM to extend the PCR indicated by EfiTcgEvent.Header.PCRIndex with the measurement digest. If the command cannot be sent successfully, the function SHALL return EFI_DEVICE_ERROR. Firmware SHALL extend digests for all active PCR banks.

   Note: firmware may use TPM2_PCR_Event to send the DataToHash to the TPM, which will compute the hash for all active PCR banks and return the respective digests. If DataToHash does not fit into a TPM2_PCR_Event the commands TPM2_HashSequenceStart, TPM2_SequenceUpdate, and TPM2_EventSequenceComplete can be used.
8. If a previous call to this function returned EFI_VOLUME_FULL and the EFI_TCG2_EXTEND_ONLY bit is set in the Flags parameter, the function SHALL return EFI_VOLUME_FULL. (No attempt is made to add the event log entry to the event log(s).)

9. For a SHA1 log format (EFI_TCG2_EVENT_LOG_FORMAT_TCG_1_2), the function SHALL build a TCG event log entry as follows: (Note: The TCG_PCR_EVENT structure SHALL be considered byte-aligned.)
   a. TCG_PCR_EVENT.PCRIndex = EfiTcgEvent.Header.PCRIndex
   b. TCG_PCR_EVENT.EventType = EfiTcgEvent.Header.EventType
   c. TCG_PCR_EVENT.Digest = <the SHA1 measurement digest calculated above>
   d. TCG_PCR_EVENT.EventSize = EfiTcgEvent.Size – sizeof(UINT32) - EfiTcgEvent.Header.HeaderSize
   e. TCG_PCR_EVENT.Event = EfiTcgEvent.Event (Note: this is a memory copy of EventSize bytes)
10. For a crypto agile log format (EFI_TCG2_EVENT_LOG_FORMAT_TCG_2), the function SHALL build a TCG event log entry as follows:
   a. TCG_PCR_EVENT2.PCRIndex = EfiTcgEvent.Header.PCRIndex
   b. TCG_PCR_EVENT2.EventType = EfiTcgEvent.Header.EventType
   c. TCG_PCR_EVENT2.Digests.Count = number of digest, i.e. active PCR banks
   d. For each active PCR bank:
      i. TCG_PCR_EVENT2.Digests[currentPcrBank].algId = algorithm ID of that PCR bank
      ii. TCG_PCR_EVENT2.Digests[currentPcrBank].digest = <digests according to the currently active PCR bank >

Note: The digests SHALL appear in the same order in all events.

Note: Because the digest values are of different size and the event structure is densely packed, the used array notation is symbolic and should be replaced with a correct offset calculation in the implementation.

   e. TCG_PCR_EVENT2.EventSize = EfiTcgEvent.Size – sizeof(UINT32) - EfiTcgEvent.Header.HeaderSize
   f. TCG_PCR_EVENT2.Event = EfiTcgEvent.Event (Note: this is a memory copy of EventSize bytes)
11. The function MAY build similar event log entries for other supported event log formats.
12. If the event log entry created above does not fit in the area allocated for the TCG log, the function SHALL return EFI_VOLUME_FULL.
13. If the firmware supports additional event log formats and any of the events created for those event logs would exceed the area allocated for the event log, the function SHALL return EFI_VOLUME_FULL.
14. The function SHALL append the events created to their corresponding event logs and the service SHALL update its internal pointer to the start of the last event for each event log.

The description of the construction of the event log is explicit to clearly define expected behavior. Other implementation that provide the same behavior at the protocol level are acceptable.

6.6.6 Status Codes Returned

Table 12: HashLogExtendEvent Return Values

Return Code	Description
EFI_SUCCESS	Operation completed successfully.
EFI_DEVICE_ERROR	The command was unsuccessful.
EFI_VOLUME_FULL	The extend operation occurred, but the event could not be written to one or more event logs.
EFI_INVALID_PARAMETER	One or more of the parameters are incorrect.
EFI_UNSUPPORTED	The PE/COFF image type is not supported.

6.7 EFI_TCG2_PROTOCOL.SubmitCommand

This service enables the sending of commands to the TPM.

6.7.1 Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG2_SUBMIT_COMMAND) (
  IN EFI_TCG2_PROTOCOL   *This,
  IN UINT32              InputParameterBlockSize,
  IN UINT8               *InputParameterBlock,
  IN UINT32              OutputParameterBlockSize,
  IN UINT8               *OutputParameterBlock
);
```

6.7.2 Parameters

Table 13: SubmitCommand Parameters

Parameter	Description
This	Indicates the calling context.
InputParameterBlockSize	Size of the TPM input parameter block.
InputParameterBlock	Pointer to the TPM input parameter block.
OutputParameterBlockSize	Size of the TPM output parameter block.

Parameter	Description
OutputParameterBlock	Pointer to the TPM output parameter block.

6.7.3 Description

The EFI_TCG2_PROTOCOL Submit Command function call provides a pass-through capability from the caller to the system's TPM.

The caller is responsible for building the command byte-stream to be sent to the TPM and is also responsible for interpreting the resulting byte-stream returned by the TPM. The TPM in and out operands for each TPM command are defined elsewhere.

Note that the returned status codes reflect the outcome of the function invocation and not the success (or failure) of the underlying TPM command.

The firmware SHALL not return TPM2_RC_RETRY prior to the completion of the call to ExitBootServices().

Implementer's Note: the implementation of this function should check the return value in the TPM response and, if it is TPM2_RC_RETRY, resend the command. The implementation may abort if a sufficient number of retries has been done.

6.7.4 Status Codes Returned

Table 14: SubmitCommand Return Values

Return Code	Description
EFI_SUCCESS	The command byte stream was successfully sent to the device and a response was successfully received.
EFI_DEVICE_ERROR	The command was not successfully sent to the device or a response was not successfully received from the device.
EFI_INVALID_PARAMETER	One or more of the parameters are incorrect.
EFI_BUFFER_TOO_SMALL	The output parameter block is too small.

6.8 EFI_TCG2_PROTOCOL.GetActivePcrBanks

This service returns the currently active PCR banks.

6.8.1 Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG2_GET_ACTIVE_PCR_BANKS) (
  IN EFI_TCG2_PROTOCOL   *This,
  OUT UINT32             *ActivePcrBanks
);
```

6.8.2 Parameters

Table 15: GetActivePcrBanks Parameters

Parameter	Description
This	Indicates the calling context
ActivePcrBanks	Pointer to the variable receiving the bitmap of currently active PCR banks.

6.8.3 Description

The EFI_TCG2_PROTOCOL.GetActivePcrBanks function call returns the bitmap of currently active PCR banks (see explanation in Section 5.4). Values in this bitmap SHALL be from the `EFI_TCG2_BOOT_HASH_ALG_*` list of values. The returned bitmap is the same as the ActivePcrBanks field in the `EFI_TCG2_BOOT_SERVICE_CAPABILITY` structure.

6.8.4 Status Codes Returned

Table 16: GetActivePcrBanks Return Value

Return Code	Description
EFI_SUCCESS	The bitmap of active PCR banks was stored in the ActivePcrBanks parameter.
EFI_INVALID_PARAMETER	One or more of the parameters are incorrect.

6.9 EFI_TCG2_PROTOCOL.SetActivePcrBanks

This service sets the currently active PCR banks.

6.9.1 Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG2_SET_ACTIVE_PCR_BANKS) (
  IN EFI_TCG2_PROTOCOL   *This,
  IN UINT32              ActivePcrBanks
);
```

6.9.2 Parameters

Table 17: SetActivePcrBanks Parameters

Parameter	Description
This	Indicates the calling context
ActivePcrBanks	Bitmap of the requested active PCR banks. At least one bit SHALL be set.

6.9.3 Description

This function determines first if the requested bitmap of PCR banks is valid. A valid bitmap of active PCR banks is a subset of EFI_TCG2_BOOT_SERVICE_CAPABILITY.HashAlgorithmBitmap and has at least one bit and at most EFI_TCG2_BOOT_SERVICE_CAPABILITY.NumberOfPcrBanks bits set. If the requested bitmap is invalid, this function call SHALL return EFI_INVALID_PARAMETER.

If the requested bitmap of PCR banks is valid, the function compares it to the currently active PCR banks. If the bitmaps are the same, the function returns EFI_SUCCESS.

If the requested bitmap differs from the currently active PCR bitmap, the function call stores a request to change the active PCR banks in a location where it can be read on the next boot. Refer to the TCG PPI Specification for examples on how to handle such requests. The function call returns EFI_SUCCESS.

Subsequent calls to SetActivePcrBanks in the same boot cycle will overwrite previously stored values. Consider the following case: The currently active PCR bank is SHA1. An invocation of SetActivePcranks requests a change to SHA256. A subsequent invocation of SetActivePcrBanks with argument SHA1 will effectively erase the previous request. Because on reboot the currently active PCR bank is SHA1, there will be no change to the PCR banks.

When EFI_SUCCESS is returned, the caller should reboot the machine. A change of the active PCR banks only takes effect on the next reboot. The caller should only invoke SetActivePcrBanks() if a change of the PCR banks is required.

When the caller initiates the reboot, firmware will use the platform authorization value to call TPM2_PCR_Allocate() to change the allocation of the active PCR banks. Firmware has to reboot again to allow the change of PCR banks in the TPM to take effect. If the operation was successful, the EFI_TCG2_BOOT_SERVICE_CAPABILITY.ActivePcrBanks will reflect the change of PCR banks.

6.9.4 Status Codes Returned

Table 18: SetActivePcrBanks Return Values

Return Code	Description
EFI_SUCCESS	The bitmap in ActivePcrBank parameter is already active.

Return Code	Description
EFI_INVALID_PARAMETER	One or more of the parameters are incorrect.

6.10 EFI_TCG2_PROTOCOL.GetResultOfSetActivePcrBanks

This service retrieves the result of a previous invocation of SetActivePcrBanks.

6.10.1 Prototype

```
typedef
EFI_STATUS
(EFIAPI * EFI_TCG2_GET_RESULT_OF_SET_ACTIVE_PCR_BANKS) (
  IN EFI_TCG2_PROTOCOL   *This,
  OUT UINT32             *OperationPresent,
  OUT UINT32             *Response
);
```

6.10.2 Parameters

Table 19: GetResultOfSetActivePcrBanks Parameters

Parameter	Description
This	Indicates the calling context
OperationPresent	Non-zero value to indicate a SetActivePcrBank operation was invoked during the last boot.
Response	The response from the SetActivePcrBank request.

6.10.3 Description

The invocation of EFI_TCG2_PROTOCOL.SetActivePcrBanks requires two reboots of the system: The caller of EFI_TCG2_PROTOCOL.SetActivePcrBanks has to initiate a reboot, so firmware can perform the required action when platform authorization for the TPM is present. When firmware boots after a call to EFI_TCG2_PROTOCOL.SetActivePcrBanks, it will attempt to change the PCR bank(s) in the TPM using the TPM command TPM2_PCR_Allocate. Firmware should store the result of that operation. Firmware has to reboot again, because changes initiated by the TPM2_PCR_Allocate command only take effect after the TPM power cycles. Now firmware is able to return the result of EFI_TCG2_PROTOCOL.SetActivePcrBanks operation.

EFI_TCG2_PROTOCOL.GetResultOfActivePcrBanks needs two output values: one is to signal if there has been a request at all - OperationPresent. If it has a non-zero value, then the Response output value contains the result of the operation. The result of the

operation may contain values indicating that a user rejected the operation, or the TPM error code, when the TPM operation was executed but returned an error. If the Response output value is zero, the operation succeeded.

An error free sequence would have the following steps:

1. Firmware launches the OS boot-loader – firmware stays present.
2. Boot-loader invokes EFI_TCG2_PROTOCOL.SetActivePcrBanks, which stores the request, possibly in a UEFI variable. The function may perform some initial checks before setting the UEFI variable and returning a response. For instance, if the requested PCR banks are the same as the currently active PCR banks, it may return EFI_SUCCESS without storing the request in the UEFI variable. Or if the requested bitmap has no PCR bank set, SetActivePcrBanks returns EFI_INVALID_PARAMETER without storing the request.
3. Boot-loader initiates reboot.
4. Firmware checks the UEFI variable that has been set in step 2.
5. If a request is stored in the UEFI variable, firmware does:
    a. Perform sanity checks on the request, if necessary.
    b. Clear UEFI variable.
    c. Display a user PPI confirmation screen according to PPI flag settings.
    d. If user rejects, store response and go to step 6.
    e. If user confirms PPI request, execute TPM2_PCR_Allocate command.
    f. Store TPM response.
    g. Reboot.
6. Firmware launches OS boot-loader – firmware stays present.
7. Boot-loader calls EFI_TCG2_PROTOCOL.GetResultOfSetActivePcrBank and retrieves response from 5.d or 5.f. See below for possible values.
8. On the next reboot the response gets deleted.

Firmware SHALL store the response of an invocation of SetActivePcrBanks in a location that is available at least as long as the EFI_TCG2_PROTOCOL is available and return this stored response on each invocation of GetResultOfSetActivePcrBanks.

The use of EFI_TCG2_PROTOCOL.SetActivePcrBanks is similar to the use of a physical presence interface request as specified in the *TCG Physical Presence Interface Specification*. Refer to the TCG PPI specification on how to handle user interaction to approve such a change and storage of request and response.

EFI_TCG2_PROTOCOL.GetResultOfSetActivePcrBanks should set OperationPresent and Response to values defined for function “Return TPM Operation Response to OS Environment” in the TCG PPI specification. To allow reuse of existing implementations, OperationPresent can be the value of the respective PPI operation (23). Response should use the following values:

0: Success

0x00000001..0x00000FFF: Corresponding TPM error code

0xFFFFFFF0: User Abort or timeout of dialog

0xFFFFFFF1: Firmware Failure

One possible, valid implementation reuses the storage location, e.g. UEFI variable, used to store PPI requests and responses for the full OS that uses the ACPI PP interface. A sequence of steps could be:

1. The OS request a PPI operation through the ACPI PP interface. The request is stored in the single request location.
2. The OS reboots.
3. Firmware boots and detects a request.
4. Firmware acts on the request.
5. Firmware stores the response.
6. Firmware loads OS boot-loader.
7. The boot-loader request a change of PCR banks and this request is stored in the single request location.
8. Boot-loader reboots.
9. Firmware boots and detects the request.
10. Firmware acts on the request. (If firmware determines that no action should be performed, the response location is not modified.)
11. Firmware stores the response. It overwrites the response from the OS. Because the response contains the requested operation plus arguments, the OS will be able to detect if the response is actually for its request. This is a valid scenario, as PPI covers this as a dual-boot scenario.
12. Firmware (reboots and) loads the OS boot-loader.
13. The boot-loader checks the response.
14. Boot-loader loads OS.
15. OS checks PPI response from boot-loader and detect different request or different request parameters.

Another valid firmware implementation may store the request and response for the EFI_TCG2_PROTOCOL in a different location from the location for PPI request and response. This should not create conflicts, but care should be taken as to which response is presented to which caller. For example:

1. The OS request a PPI operation through the ACPI PP interface. The request is stored in the PPI specific location.
2. The OS reboots.
3. Firmware boots and detects a request in the PPI specific request location.
4. Firmware acts on the PPI request.
5. Firmware stores the response in the PPI specific location.
6. Firmware loads the OS boot-loader.
7. The boot-loader requests a change of PCR banks and this request is stored in an EFI_TCG2_PROTOCOL specific location.
8. Boot-loader reboots.

9. Firmware checks the EFI_TCG2_PROTOCOL specific request location (after not finding something in the PPI specific location) and performs the request action.
10. Firmware stores the response in the EFI_TCG2_PROTOCOL specific location.
11. Firmware (reboots and) loads the OS boot-loader.
12. The boot-loader checks the EFI_TCG2_PROTOCOL specific response.
13. Boot-loader loads OS.
14. OS checks PPI specific response.

If firmware implements the PPI ACPI protocol and stores PPI request in the same location as request from EFI_TCG2_PROTOCOL, it SHALL store the response for the remainder of the boot cycle.

6.10.4 Status Codes Returned

Table 20: GetResultOfSetActivePcrBanks Return Values

Return Code	Description
EFI_SUCCESS	The result value could be returned.
EFI_INVALID_PARAMETER	One or more of the parameters are incorrect.

7 Log entries after Get Event Log service

This table may be documented in future versions of the UEFI Specification.

All events generated after the invocation of **EFI_TCG2_GET_EVENT_LOG** SHALL be stored in an instance of an **EFI_CONFIGURATION_TABLE** named by the ***VendorGuid*** of **EFI_TCG2_FINAL_EVENTS_TABLE_GUID**, defined by:

GUID –

```
#define EFI_TCG2_FINAL_EVENTS_TABLE_GUID \
   { 0x1e2ed096, 0x30e2, 0x4254, \
      { 0xbd, 0x89, 0x86, 0x3b, 0xbe, 0xf8, 0x23, 0x25 } }
```

The associated table contents SHALL be referenced by the ***VendorTable*** of **EFI_TCG2_FINAL_EVENTS_TABLE**:

```
typedef struct tdEFI_TCG2_FINAL_EVENTS_TABLE {
   UINT64          Version;
   UINT64          NumberOfEvents;
   TCG_PCR_EVENT2  Event[NumberOfEvents];
} EFI_TCG2_FINAL_EVENTS_TABLE;


#define EFI_TCG2_FINAL_EVENTS_TABLE_VERSION   1
```

Table 21: Fields for the EFI_TCG2_FINAL_EVENTS_TABLE

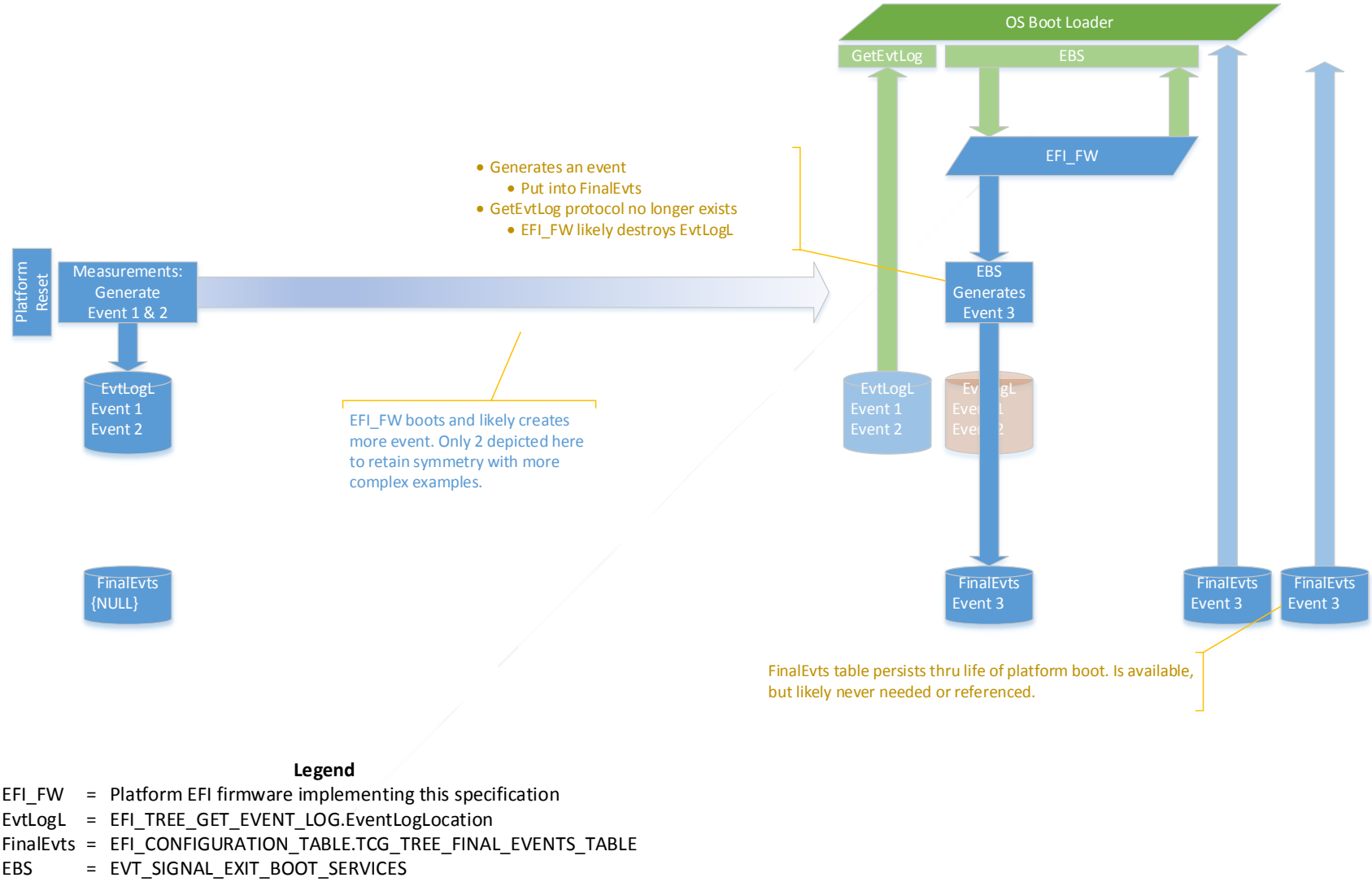
Field	Description
Version	The version of this structure. Versioning allows for possibly appending new fields at end in the future.
NumberOfEvent	Number of events recorded after invocation of GetEventLog API
Event	List of events of type TCG_PCR_EVENT2. There are NumberOfEvents events in that list.

See the UEFI 2.4 (Errata B) specification for guidance on the runtime accessible memory types for these tables and the service used to install these tables.

7.1 Event Log Retrieval Sequence

The following sequence diagrams depict the potential flow of the TCG log.

7.1.1 Minimal Options Implemented

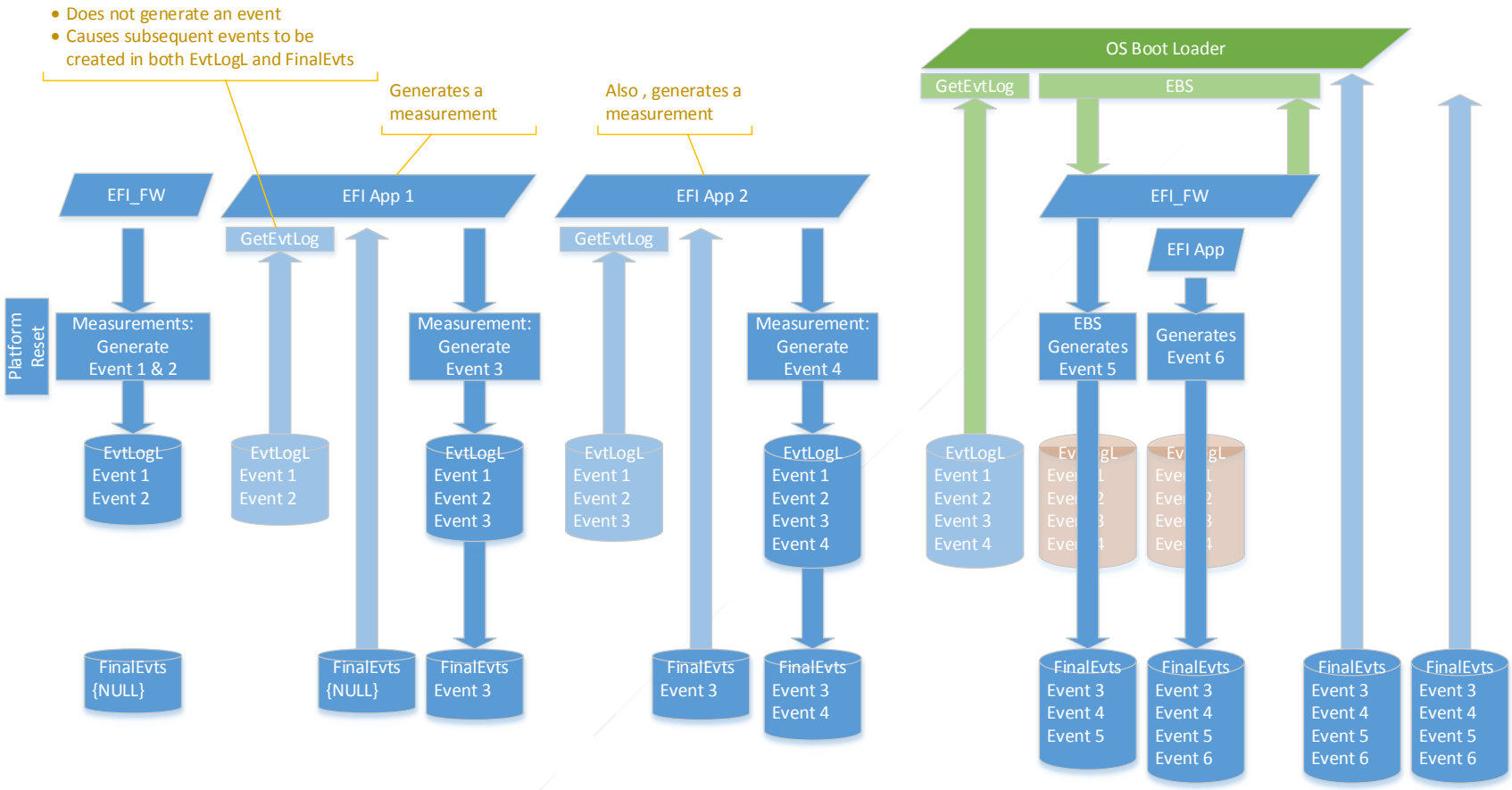


Legend

EFI_FW	=	Platform EFI firmware implementing this specification
EvtLogL	=	EFI_TREE_GET_EVENT_LOG.EventLogLocation
FinalEvts	=	EFI_CONFIGURATION_TABLE.TCG_TREE_FINAL_EVENTS_TABLE
EBS	=	EVT_SIGNAL_EXIT_BOOT_SERVICES

Figure 1: Flow diagram with minimal flow to retrieve event log

7.1.2 All Options Implemented



Legend

EFI_FW	=	Platform EFI firmware implementing this specification
EvtLogL	=	EFI_TREE_GET_EVENT_LOG.EventLogLocation
FinalEvts	=	EFI_CONFIGURATION_TABLE.TCG_TREE_FINAL_EVENTS_TABLE
EBS	=	EVT_SIGNAL_EXIT_BOOT_SERVICES

Figure 2: Flow diagram exercising all options to retrieve event log