TCG EFI Protocol Specification
For TPM Family 1.1 or 1.2

Specification Version 1.22
Revision 5
27 January 2014

Contact: admin@trustedcomputinggroup.org

TCG Published

TCG

Disclaimers, Notices, and License Terms

Acknowledgements

The writing of a specification, particularly a security specification, takes many hours for both development and review. This specification is no exception with roughly 100 individuals involved in the process. The TCG would like to acknowledge the contribution of those individuals (listed below) and the companies who allowed them to volunteer their time to the development of this specification.

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

The TCG would also like to give special thanks to Vincent Zimmer and Lee Rosenbaum who were the editors of this specification.

Contributors:

Gary Simpson; AMD

Günter Fuchs, Atmel

Bill Jacobs; Cisco

Amy Nelson; Dell

Carey Huscroft; Hewlett Packard

Lan Wang; Hewlett Packard

Graeme Proudler; Hewlett Packard

Ken Goldman, IBM

Shiva Desari, IBM

Vincent Zimmer; Intel

Lee Rosenbaum; Intel

Long Qin; Intel

Jiewen Yao; Intel

Quo Dong; Intel

Monty Wiseman; Intel

Robert Hart, Johns Hopkins University, Applied Physics Lab

Randy Springfield; Lenovo

Rahul Verma; Microsoft

David Wooten; Microsoft

Eugene Samsonov, Microsoft

Dick Wilkens; Phoenix

Greg Kazmierczak; Wave Systems

Tom Brostrom, United States Government

Table of Contents

1. Introduction

Start of informative comment:

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 of the three example instantiations of an EFI platform shown in Figures 2-1, 2-2, and 2-3, as well as other instantiations.

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

This specification complements the EFI 1.10 Specification.

This TCG EFI Protocol Specification is a pure interface specification that provides no information on "how" to construct the underlying firmware implementation. For that information, see the specifications referenced in Section 1.2, External Specifications.

End of informative comment.

1.1 Conventions Used in this Document

Start of informative comment:

This section gives the data structure description and typographic conventions used in this document.

End of informative comment.

1.1.1 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.

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.

1.1.2 Typographic Conventions

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

`Prototype`	This typeface indicates prototype code.
`Argument`	This typeface indicates arguments.
`Name`	This typeface indicates actual code or a programming construct.
register	This typeface indicates a processor register.

1.2 External Specifications

References to external specifications:

Extensible Firmware Interface Main Specification Version 1.10, http://www.intel.com/technology/efi

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

TCG TPM Main Specification Version 1.2, https://www.trustedcomputinggroup.org/specs/TPM

TCG PC Client Implementation Specification for Conventional BIOS, Version 1.2, *https://www.trustedcomputinggroup.org/specs/PCClient/*

TCG EFI Platform Specification, Version 1.2, *https://www.trustedcomputinggroup.org/specs/PCClient/*

TCG Itanium Architecture Server Specification, Version 1.0, *https://www.trustedcomputinggroup.org/specs/Server/*

1.3 Abbreviations and Terminology

This specification uses the following abbreviations:

Boot Services
(This definition is copied and pasted from the EFI 1.10 Specification, for the convenience of the reader) The collection of interfaces and protocols present in the boot environment. These services minimally provide an OS loader with access to platform capabilities required to complete OS boot. [Boot] services are also available to drivers and applications that need access to platform capability. Boot services terminate once the operating system takes control of the platform.

Boot Services Time
(This definition is copied and pasted from the EFI 1.10 Specification, for the convenience of the reader) The period between platform initialization and the call to **ExitBootServices ( )**. During this time, EFI drivers and applications are loaded iteratively as 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 1.10 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 an EFI driver 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 1.10 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 1.10 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.

ESP
EFI System Partition: Portion of the hard disk reserved for EFI. It is formatted as FAT12, 16, or 32.

Event Services

The set of functions used to manage EFI events. Includes **CheckEvent ( )**, **CreateEvent ( )**, **CloseEvent ( )**, **SignalEvent ( )**, and **WaitForEvent ( )**.

GPT
GUID'd Partition Table: Partitioning scheme that subsumes MBR and uses GUIDs and a new table definition to describe regions of disk.

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

Image
An executable file stored in a file system that complies with the EFI 1.10 Specification. One type of image is a driver.

Image Handle
(This definition is copied and pasted from the EFI 1.10 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 1.10 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
EFI Interface: Includes methods and instance data, similar to the public portion of a C++ Class.

System Table
The EFI System Table contains pointers to the active console devices, a pointer to the EFI Boot Services Table, a pointer to the EFI Runtime Services Table, and a pointer to a list of system configuration tables, such as ACPI, SMBIOS, and the SAL System Table. For more information about the EFI System Table, see section 4.1 of the Extensible Firmware Specification, version 1.10, December 1, 2002.

TPM
Trusted Platform Module

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

2. Implementation Overview

Start of informative comment:

There are various embodiments of EFI; this section shows three such embodiments.

End of informative comment.

2.1 EFI

Start of informative comment:

Figure 2-1 shows an EFI Client platform.

The TCG EFI protocol defined in this specification shows as a dark, horizontal line.

End of informative comment.

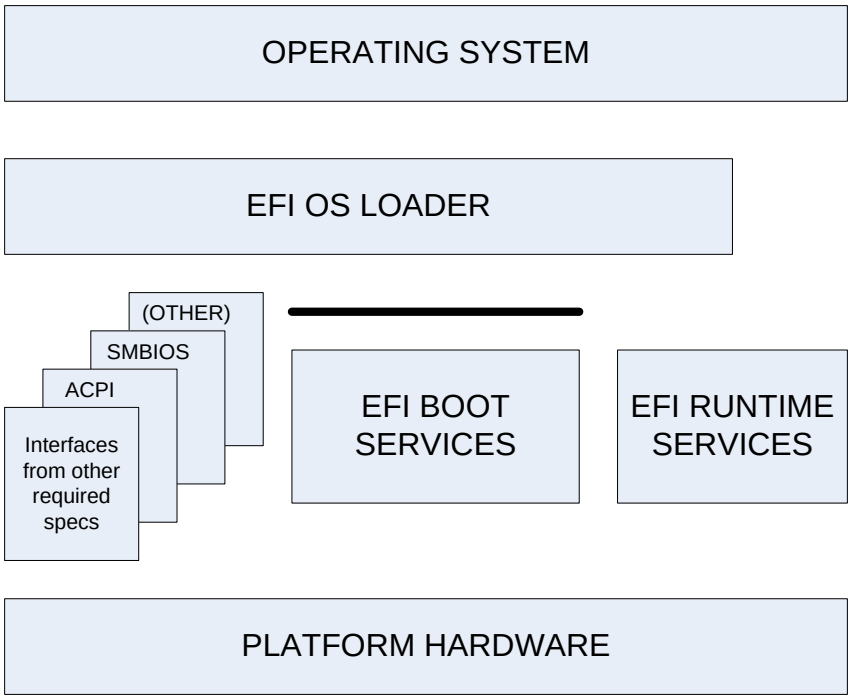

Figure 2.1 Native EFI Platform

2.2 EFI on Top of Conventional BIOS

Start of informative comment:

Figure 2-2 shows EFI on top of conventional BIOS.

Note that Figure 2-2 shows a series of drivers and a core infrastructure inside the box labeled "EFI Boot Services." One or more of the series of the Boot Services Drivers may be an EFI 1.10 driver that abstracts and provides an instance of the EFI_TCG Protocol. This specification primarily documents the EFI_TCG Protocol that such a driver uses to communicate with the EFI Boot Services.

For a reference to more information about EFI, see section 1.2, External Specifications.

End of informative comment.

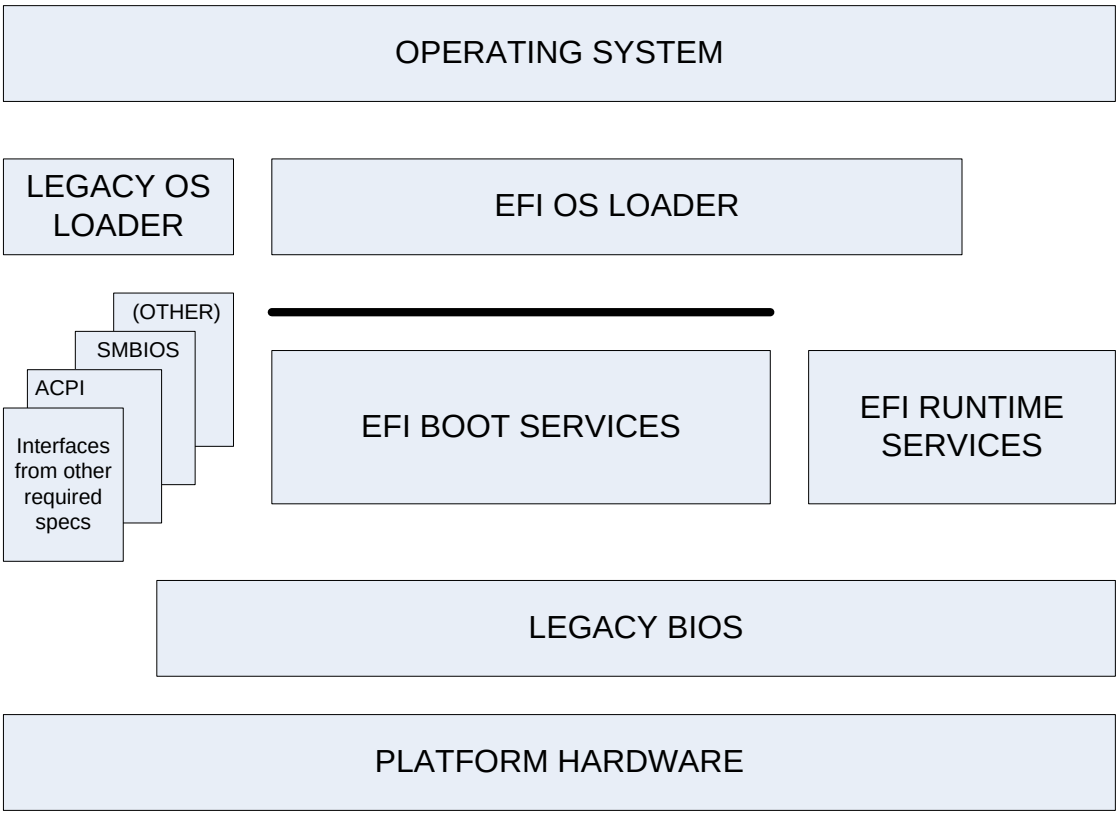

Figure 2.2 EFI on Top of Conventional BIOS

2.3 EFI Layered on Top of SAL

Start of informative comment:

Another embodiment of EFI is EFI on top of the System Abstraction Layer (SAL). Figure 2.3, below, shows this layering.

For a reference to more information about SAL, see section 1.2, External References.

End of informative comment.

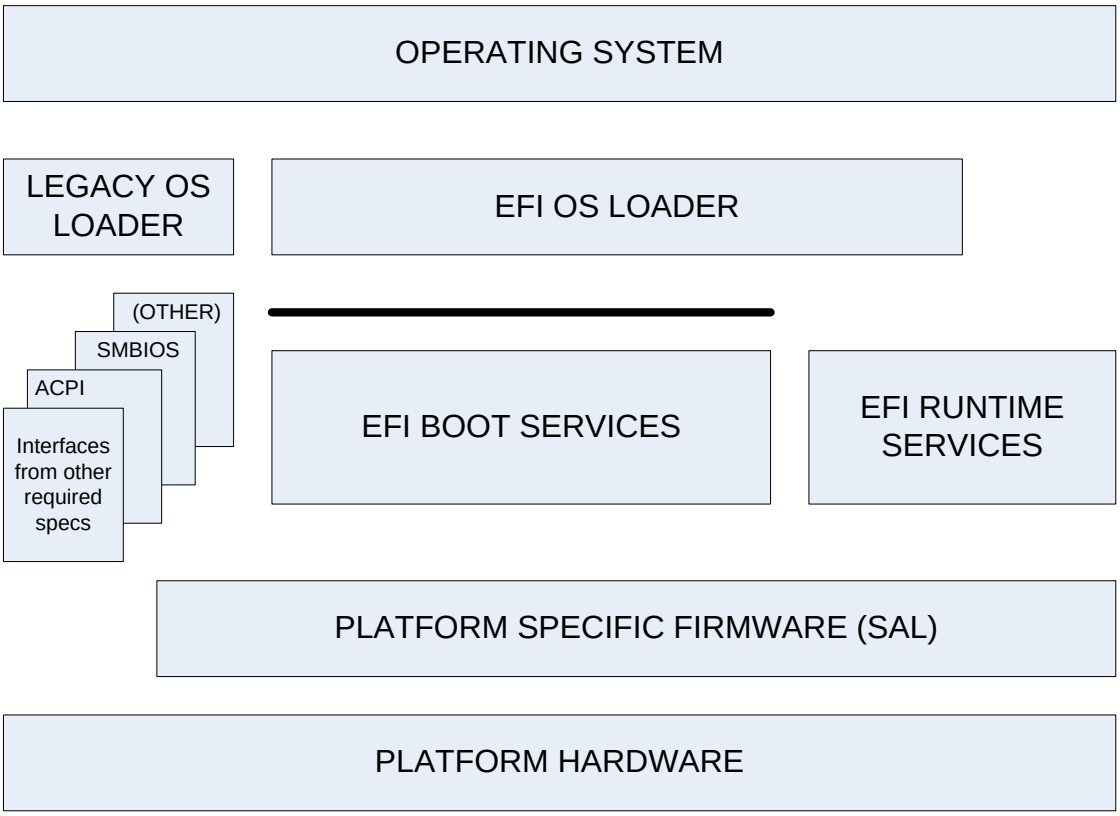

Figure 2.3 EFI Layered on Top of System Abstraction Layer (SAL)

3. TCG Protocol

Start of informative comment:

This section defines the TCG Protocol that abstracts the TCG policy services, the **`EFI_TCG`** protocol.

End of informative comment.

3.1 EFI_TCG Protocol

This section provides a detailed description of the **EFI_TCG** protocol.

GUID

```
#define EFI_TCG_PROTOCOL_GUID   \
      {0xf541796d, 0xa62e, 0x4954, 0xa7, 0x75, 0x95, 0x84, 0xf6,
      0x1b, 0x9c, 0xdd}
```

Protocol Interface Structure

```
typedef struct {
  EFI_TCG_STATUS_CHECK              StatusCheck;
  EFI_TCG_HASH_ALL                  HashAll;
  EFI_TCG_LOG_EVENT                 LogEvent;
  EFI_TCG_PASS_THROUGH_TO_TPM       PassThroughToTPM;
  EFI_TCG_HASH_LOG_EXTEND_EVENT     HashLogExtendEvent;
} EFI_TCG_PROTOCOL;
```

Parameters

StatusCheck	This service provides information on the TPM.
HashAll	This service abstracts the capability to do a hash operation on a data buffer.
LogEvent	This service abstracts the capability to add an entry to the Event Log.
PassThroughToTPM	This service provides a pass-through capability from the caller to the system's TPM.
HashLogExtendEvent	This service abstracts the capability to do a hash operation on a data buffer, extend a specific TPM PCR with the hash result, and add an entry to the Event Log.

Description

The **EFI_TCG** Protocol abstracts TCG 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. The OS must have its own TPM 1.2 driver loaded and ready when ExitBootServices ( ) is called. The OS cannot use TPM services that reside in motherboard firmware after ExitBootServices ( ) successfully completes.

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 ( )** cannot clean up on behalf of drivers that have been loaded into the system. The drivers have to do that for themselves by creating an event whose type is EVT_SIGNAL_EXIT_BOOT_SERVICES and whose notification function is a function within the driver itself. Then, when **ExitBootServices ( )** has finished its cleanup, it signals each 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 5 of the EFI 1.10 Specification.

3.1.1 EFI_TCG_PROTOCOL.StatusCheck()

Summary

This service provides EFI protocol capability information, state information about the TPM, and Event Log state information.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG_STATUS_CHECK) (
  IN struct _EFI_TCG_PROTOCOL            *This,
  OUT TCG_BOOT_SERVICE_CAPABILITY        *ProtocolCapability,
  OUT UINT32                             *TCGFeatureFlags,
  OUT EFI_PHYSICAL_ADDRESS               *EventLogLocation
  OUT EFI_PHYSICAL_ADDRESS               *EventLogLastEntry
  );
```

Parameters

This	Indicates the calling context; see Section 3.1 for the definition of the **EFI_TCG_PROTOCOL** type.
ProtocolCapability	The callee allocates memory for a TCG_BOOT_SERVICE_CAPABILITY structure and fills in the fields with the EFI protocol capability information and the current TPM state information.
TCGFeatureFlags	This is a pointer to the feature flags. No feature flags are currently defined so this parameter MUST be set to 0. However, in the future, feature flags may be defined that, for example, enable hash algorithm agility.
EventLogLocation	This is a pointer to the address of the event log in memory.
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. See the Description section for information about values returned in this parameter in the special cases of an empty Event Log or an Event Log with only one entry..

Related Definitions

```
typedef struct {
  UINT8  Major;
  UINT8  Minor;
  UINT8  RevMajor;
  UINT8  RevMinor;
} TCG_VERSION;

typedef    UINT64     EFI_PHYSICAL_ADDRESS

typedef struct {
  UINT8      Size;                //Size of this structure
```

```
  TCG_VERSION StructureVersion;
  TCG_VERSION ProtocolSpecVersion;
  UINT8       HashAlgorithmBitmap   //Hash algorithms this
                                    //protocol is capable of:
                                    // 01=SHA-1
  BOOL    TPMPresentFlag            //00h=TPM not present
  BOOL    TPMDeactivatedFlag;       //01h=TPM currently
                                    //deactivated
} TCG_EFI_BOOT_SERVICE_CAPABILITY;
```

Description

The **EFI_TCG_PROTOCOL** Status Check function call provides EFI protocol capability information, state information about the TPM, and Event Log state information.

The caller does not use the Status Check function call to verify the presence of the TCG EFI protocol. On an EFI-capable platform that is done by LocateHandle / Open Protocol.

When the caller invokes the **EFI_TCG_PROTOCOL** Status Check function, the Event Log may be empty, may have exactly one entry, or may contain more than one entry.

- If the Event Log is empty, the *EventLogLastEntry* parameter value MUST be 0 (zero)
- If the Event Log contains exactly one entry, the *EventLogLastEntry* parameter value MUST equal the value of the *EventLogLocation* parameter
- If the EventLog contains more than one entry, the *EventLogLastEntry* parameter value MUST be a pointer to the address of the start of the last entry in the event log in memory

.

Status Codes Returned

EFI_SUCCESS	Operation completed successfully.
EFI_INVALID_PARAMETER	***ProtocolCapability*** does not match TCG capability.

.

3.1.2 EFI_TCG_PROTOCOL.HashAll()

Summary

This service abstracts the capability to do a hash operation on a data buffer.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG_HASH_ALL) (
  IN struct _EFI_TCG_PROTOCOL    *This,
  IN UINT8                       *HashData,
  IN UINT64                      HashDataLen,
  IN TCG_ALGORITHM_ID            AlgorithmId,
  IN OUT UINT64                  *HashedDataLen,
  IN OUT UINT8                   **HashedDataResult
);
```

Parameters

This	Indicates the calling context; see Section 3.1 for the definition of the **EFI_TCG_PROTOCOL** type.
HashData	Pointer to the data buffer to be hashed
HashDataLen	Length of the data buffer to be hashed
AlgorithmId	Identification of the Algorithm to use for the hashing operation
HashedDataLen	Resultant length of the hashed data
HashedDataResult	Resultant buffer of the hashed data

Related Definitions

```
typedef UINT32  TCG_ALGORITHM_ID;
#define TCG_ALG_SHA  0x00000004  // The SHA1 algorithm
```

Description

The **EFI_TCG_PROTOCOL** Hash All Interface function performs the required hash operation on the input data and returns the resulting hash to the caller.

Status Codes Returned

EFI_SUCCESS	Operation completed successfully.
EFI_INVALID_PARAMETER	***HashDataLen*** is NULL incorrect.
EFI_INVALID_PARAMETER	***HashDataLenResult*** is NULL
EFI_OUT_OF_RESOURCES	Cannot allocate buffer of size ****HashedDataLen***
EFI_UNSUPPORTED	AlgorithmId not supported
EFI_BUFFER_TOO_SMALL	*HashedDataLen < sizeof (TPM_DIGEST)

3.1.3 EFI_TCG_PROTOCOL.LogEvent()

Summary

This service abstracts the capability to add an entry to the Event Log.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG_LOG_EVENT) (
  IN struct _EFI_TCG_PROTOCOL                *This,
  IN TCG_PCR_EVENT                           *TCGLogData,
  IN OUT UINT32                              *EventNumber,
  IN UINT32                                  Flags
);
```

Parameters

This	Indicates the calling context; see Section 3.1 for the definition of the **EFI_TCG_PROTOCOL** type.
TCGLogData	Pointer to the start of the data buffer containing the TCG_PCR_EVENT data structure. All fields in this structure are properly filled by the caller.
EventNumber	The event number of the event just logged
Flags	Indicate additional flags. Only one flag has been defined at this time, which is 0x01 and means the extend operation should not be performed. All other bits are reserved.

Related Definitions

```
#define SHA1_DIGEST_SIZE   20

typedef struct {
 UINT8  Digest[SHA1_DIGEST_SIZE];
} TCG_DIGEST;

typedef TCG_DIGEST TCG_COMPOSITE_HASH;

typedef UINT32  TCG_EVENTTYPE;

//
//  Log event types
//
#define EV_NO_ACTION               3
#define EV_SEPARATOR               4
#define EV_ACTION                  5
#define EV_EVENT_TAG               6
#define EV_CPU_MICROCODE           9
#define EV_PLATFORM_CONFIG_FLAGS   10
#define EV_IPL                     13
#define EV_IPL_PARTITION_DATA      14
```

```
#define EV_NONHOST_CODE            15
#define EV_NONHOST_CONFIG          16

typedef struct {
   TCG_PCRINDEX    PCRIndex;  //PCR Index value that either
                              //matches the PCRIndex of a
                              //previous extend operation or
                              //indicates that this Event Log
                              //entry is not associated with
                              //an extend operation
   TCG_EVENTTYPE   EventType; //See Log event types defined above
   TCG_DIGEST      digest;    //The hash of the event data
   UINT32          EventSize; //Size of the event data
   UINT8           Event[1];  //The event data
} TCG_PCR_EVENT;              //Structure to be added to the
                              //Event Log
```

For the definition of the EFI platform-specific event types that are sub-types of the general event type EV_EVENT_TAG, see section 8.2 of the TCG EFI Platform Specification, version 1.2, available at http://www.trustedcomputinggroup.com/.

Description

This function performs the same operations as **EFI_TCG_HASH_LOG_EXTEND_EVENT**, except it does not perform the TPM_Extend operation.

There are two reasons to call the **TCGLogEvent** function:

(1) To add an informative entry to the Event Log that is not associated with an extend operation to a PCR. The values within such an entry cannot be verified, but the entry may serve an informative or delimiting function, as indicated by the EventType. In this case, the caller SHOULD set the value of the Flags parameter to -1 (all ones) to provide further indication that no extend operation is to be performed.

(2) To add an entry to the EventLog that is associated with a previous extend operation. The previous extend operation might have been done in an environment – for example, by the CRTM – where no memory is available to create the Event Log entry, In this case, the caller MUST set the value of the PCRIndex parameter to the PCRIndex of the PCR into which the previous extend operation was done.

Status Codes Returned

EFI_SUCCESS	Operation completed successfully.
EFI_OUT_OF_RESOURCES	Insufficient memory in the event log to complete this action

3.1.4 EFI_TCG_PROTOCOL.PassThroughToTpm()

Summary

This service is a proxy for commands to the TPM.

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG_PASS_THROUGH_TO_TPM) (
  IN struct _EFI_TCG_PROTOCOL     *This,
  IN UINT32                       TpmInputParameterBlockSize,
  IN UINT8                        *TpmInputParameterBlock,
  IN UINT32                       TpmOutputParameterBlockSize,
  IN UINT8                        *TpmOutputParameterBlock
  );
```

Parameters

This	Indicates the calling context; see Section 3.1 for the definition of the **EFI_TCG_PROTOCOL** type.
TpmInputParameterBlockSize	Size of the TPM input parameter block
TpmInputParameterBlock	Pointer to the TPM input parameter block
TpmOutputParameterBlockSize	Size of the TPM output parameter block
TpmOutputParameterBlock	Pointer to the TPM output parameter block

Description

The **EFI_TCG_PROTOCOL** Pass Through to TPM function call provides a pass-through capability from the caller to the system's TPM.

The caller's responsibilities include building the command byte-stream to be sent to the TPM and interpreting the resulting byte-stream returned by the TPM. The Main Specification defines TPM in and out operands for each TPM command. .

Status Codes Returned

EFI_SUCCESS	Operation completed successfully.
EFI_UNSUPPORTED	Current Task Priority Level >= EFI_TPL_CALLBACK
EFI_TIMEOUT	The TIS timed-out
EFI_INVALID_PARAMETER	Invalid ordinal

3.1.5 EFI_TCG_PROTOCOL.TCGHashLogExtendEvent()

Summary

This service abstracts the capability to do a hash operation on a data buffer, extend a specific TPM PCR with the hash result, and add an entry to the Event Log

Prototype

```
typedef
EFI_STATUS
(EFIAPI *EFI_TCG_HASH_LOG_EXTEND_EVENT) (
  IN struct _EFI_TCG_PROTOCOL    *This,
  IN EFI_PHYSICAL_ADDRESS        HashData,
  IN UINT64                      HashDataLen,
  IN TCG_ALGORITHM_ID            AlgorithmId,
  IN OUT TCG_PCR_EVENT           *TCGLogData,
  IN OUT UINT32                  *EventNumber
  OUT EFI_PHYSICAL_ADDRESS       *EventLogLastEntry
  );
```

Parameters

This	Indicates the calling context. Section 3.1 defines type **EFI_TCG_PROTOCOL**.
HashData	Physical address of the start of the data buffer to be hashed, extended, and logged.
HashDataLen	The length, in bytes, of the buffer referenced by HashData
AlgorithmId	Identification of the Algorithm to use for the hashing operation
TCGLogData	The physical address of the start of the data buffer containing the **TCG_PCR_EVENT** data structure.
EventNumber	The event number of the event just logged.
EventLogLastEntry	Physical address of the first byte of the entry just placed in the Event Log. If the Event Log was empty when this function was called, then this physical address will be the same as the physical address of the start of the Event Log.

Related Definitions

See section 3.1.3, EFI_TCG_PROTOCO.LogEvent, for the definitions of the event log entry structure and the general event types that use that structure..

Description

The **EFI_TCG_PROTOCOL** Hash Log Extend Event function call performs hashing of the event or the event data, extends the event to a PCR, and then places the resulting **TCG_PCR_EVENT** structure data into the Event Log.

If this function cannot create an Event Log entry (for example, because the Event Log is full), then this function MUST perform the TPM_Extend operation.

If the HashData input parameter is not NULL or the HashDataLen input parameter is not NULL, then this function MUST

1. Treat the data buffer pointed to by the HashData input parameter as a **TCG_PCR_EVENT** structure, with all fields completed except the **TCG_PCR_EVENT.digest** field
2. Perform the hash function on the **TCG_PCR_EVENT.event** field
3. Complete the **TCG_PCR_EVENT** structure by placing the resulting hash, H1, into the **TCG_PCR_EVENT.digest** field
4. Perform a TPM_Extend operation, using H1 as the input to the TPM_Extend
5. Place the completed **TCG_PCR_EVENT** structure into the Event Log
6. Increment the event number value pointed to by the *EventNumber input parameter
7. Place H1 into the data area pointed to by the **HashValue input parameter
8. Return with the appropriate Status Code

If the HashData input parameter is NULL and the HashDataLen input parameter is NULL, then this function MUST

1. Treat the data buffer pointed to by the HashData input parameter as a completed **TCG_PCR_EVENT** structure, with all fields completed including the **TCG_PCR_EVENT.digest** field
2. Perform a TPM_Extend operation using the **TCG_PCR_EVENT.digest** field as input to the TPM_Extend operation
3. Place the **TCG_PCR_EVENT** structure into the Measurement Log
4. Increment the event number value pointed to by the *EventNumber input parameter
5. Place the **TCG_PCR_EVENT.digest** field into the data area pointed to by the **HashValue input parameter
6. Return with the appropriate Status Code

Status Codes Returned

EFI_SUCCESS	Operation completed successfully.
EFI_UNSUPPORTED	AlgorithmId != TPM_ALG_SHA
EFI_UNSUPPORTED	Current TPL >= EFI_TPL_CALLBACK
EFI_DEVICE_ERROR	The command was unsuccessful.