White Paper

EDK II Topology – Trusted Platform Module (TPM)

Lee Hamel
Intel Corporation

Stephen Bekefi
Intel Corporation

January 2017

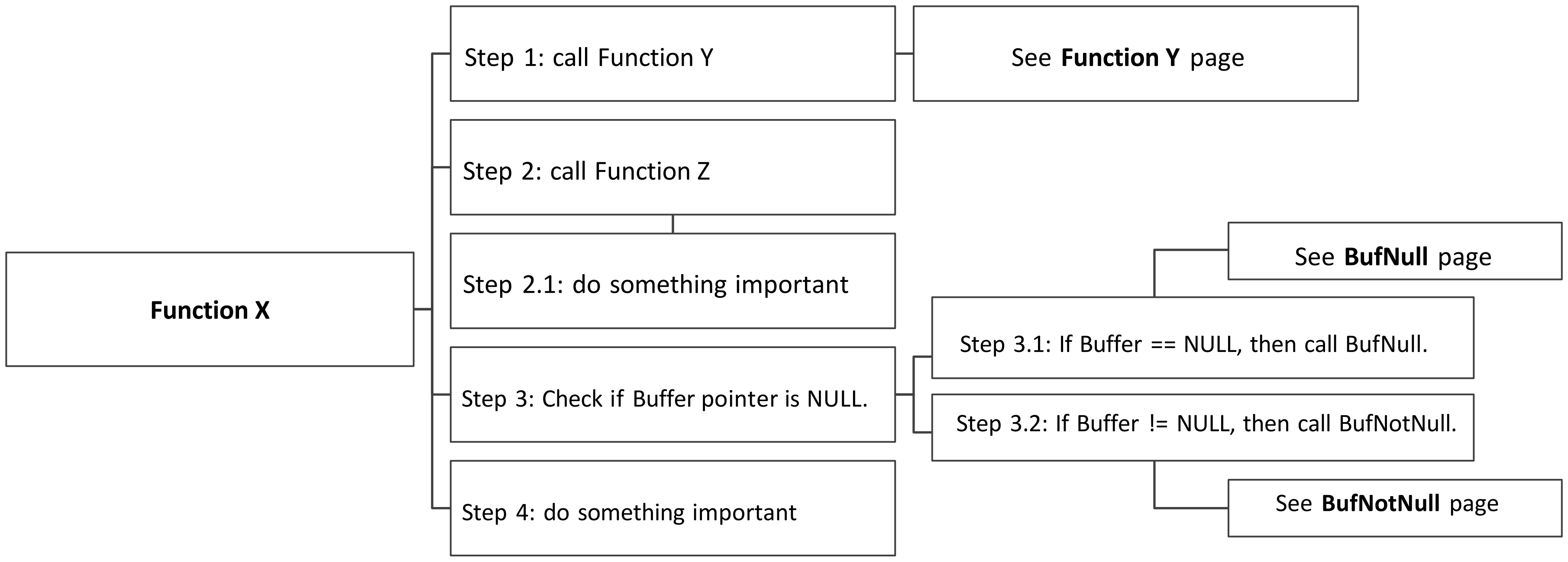
Table of Contents

Graph Interpretation

The function being discussed always starts in the only box on the far left. Boxes represent steps in a function, a branch evaluation in a function, or a note to see details on another page about a function being called at a step. Connectors between boxes indicate code flow (who called what) and should be read left-to-right, top-to-bottom. Text in each box will indicate if it is a call, a branch evaluation, or a note. This format was chosen to fit important function details on 1 page.

For the example below, the equivalent C code (note connectors from Step 1-4 to Function X):

```
FunctionX() {
FunctionY();    // see details on Function Y page
FunctionZ();    // Step 2; Step 2.1 is in FunctionZ() and is listed because it is important; note the
                   connector between Step 2 and 2.1
If (Buffer == NULL)   // Step 3
     BufNull();       // Step 3.1
else
     BufNotNull();   // Step 3.2
Step 4
}
```


Function X
Step 1: call Function Y
See **Function Y** page
Step 2: call Function Z
Step 2.1: do something important
Step 3: Check if Buffer pointer is NULL.
Step 4: do something important
See **BufNull** page
Step 3.1: If Buffer == NULL, then call BufNull.
Step 3.2: If Buffer != NULL, then call BufNotNull.
See **BufNotNull** page

Protocol Information

How protocol services are defined to find such protocol codes to examine

Summary: if one wants to examine the code for a protocol function, one should find the structure definition for that protocol. First find the declaration of the structure, then find the structure member that corresponds with the protocol in the structure definition because they may have different names. For example: `EFI_BOOT_SERVICES` defines `LocateProtocol, mBootServices` is of type `EFI_BOOT_SERVICES`, and the structure member `CoreLocateProtocol` corresponds with the structure definition `LocateProtocol`.

The code `MdePkg\Include\UefiSpec.h` defines `EFI_BOOT_SERVICES` structure, and has structure members for protocol services (`LocateProtocol, InstallProtocolInterface`, etc). The code for `MdeModulePkg\ Core\Dxe\DxeMain.c` has a variable `mBootServices` of type `EFI_BOOT_SERVICES`. The code for `mBootServices` sets function pointers for functions such as `LocateProtocol` to `CoreLocateProtocol` and `InstallMultipleProtocolInterfaces` to `CoreInstallMultipleProtocolInterfaces.` These functions are defined in the EDK II code: `MdeModulePkg\Core\Dxe\Handle.c`.

How protocols are loaded from flash into memory

Summary: Drivers are loaded from flash into memory by some mechanisms into a linked list during the PEI phase.

EDK II code: `MdeModulePkg\Core\Dxe\DxeMain.c` has the function `DxeMain` which is called when the DXE Core driver is loaded. EDK II code: `MdeModulePkg\Core\Dxe\DxeMain.inf` has the following:

```
MODULE_TYPE     = DXE_CORE
ENTRY_POINT     = DxeMain
```

The end of `DxeMain` calls the function `CoreInstallMultipleProtocolInterface` with the GUID for the HOB that was populated with drivers from the flash part during PEI. PEI phase calls `ReadSection` (associated with `FvReadFileSection` in EDEK II code for `MdeModulePkg\Universal\FirmwareVolume\FwVolDxe\FwVol.c`), which eventually gets to a call to `LocateProtocol` with `gEfiDecompressProtocolGuid` as a parameter.

About Trusted Platform Module (TPM)

Acronyms

TPM: Trusted Platform Module

SHA: Secure Hash Algorithm

PCR: Platform Configuration Register

PCD: Platform Configuration Database (EDK II code)

TPM Types and Protocols

Currently there are two TPM device types: 1.2 and 2.0. A TPM 1.2 device is implemented as a discrete device and only uses the TCG protocol (`gEfiTcgProtocolGuid`) in EDK II. Architecturally a TPM 2.0 device is very different from a TPM 1.2 device. Furthermore, there are two kinds of TPM 2.0 devices and two available protocols. A TPM 2.0 device is available as a discrete device and firmware implemented device, both accessible with the TrEE (`gEfiTrEEProtocolGuid`) and TCG2 (`gEfiTcg2ProtocolGuid`) protocols. The TrEE protocol, developed by Microsoft*, was available earlier than the TCG2 protocol, though they are essentially the same. The two protocols have the same GUID so by UEFI rules they are the same.

Using a TPM 1.2 device

The EDK II Code: `Tpm12DeviceLib.h` provides a general pass-through interface for sending the TPM commands. These commands have to be properly formatted and packed since the bytes go directly to the TPM. The function `Tpm12RequestUseTpm()` allows the user to request use of the TPM while the function `Tpm12SubmitCommand()` allows the programmer to send commands.

The EDK II Code: `Tpm12CommandLib.h` provides an easier to use interface for sending commonly used commands. Command packing is done for the programmer; the commands only require filling out function parameters for the TPM command. The interface is most useful for defining indexes, reading from them, and writing to them. It also allows for startup, shutdown, and reading TPM flags of different kinds.

The EDK II Code: `TpmCommLib.h` provides a short but slightly more advanced interface for a TPM. The EDK II Code: `CommonLib.h` header file defines how to access the TIS interface, which is how a TPM sends and receives bytes on the LPC bus. It includes the `TIS_PC_REGISTERS` which are used to communicate with a TPM on the most basic level. TPM registers such as `Access`, `BurstCount`, and `DataFifo` are essential to understanding the TPM state and sending bytes.

Using a TPM 2.0 device

The EDK II Code: `Tpm2DeviceLib.h` provides a general pass-through interface for sending the TPM commands. Even though the two types of TPMs are very different architecturally, the interface via the driver has been standardized. These commands have to be properly formatted and packed since the bytes go directly through to the TPM like in the TPM 1.2 example. The function

`Tpm2RequestUseTpm` allows the user to request use of the TPM while the function **`Tpm2SubmitCommand`** allows the programmer to send commands.

The EDK II Code: `Tpm2CommandLib.h` provides an easier to use interface for sending commonly used commands. This is especially important in TPM2 since the command packing is orders of magnitude more complex than in TPM 1.2. Commands packing are done for the programmer; the commands only require filling out function parameters for the TPM command. The available interface   provides the same basic functionality as is provided in the TPM 1.2 command lib; defining indexes, reading from them and writing to them, startup, shutdown, and reading TPM flags of different kinds.

TPM Usage

A TPM can be queried to return its capabilities, such as PCR count and supported hash types.

PCR indexes are typically used for specific purposes, e.g. **`PCR0`** is used to measure BIOS code.

A TPM is used to measure a block of memory via a SHA and extend the hash value to a PCR on the TPM device. The term   measure   means that a block of memory (code, data structure,   configuration data, etc) is hashed by a TPM SHA. The result of the TPM SHA, a hash digest or hash value or simply hash, is then extended to a PCR. The term extend is used because a PCR may have a hash from a previous measurement, and by using the same PCR the 2 hashes (current and new) are hashed and written to the PCR, thus extending the hash. This is done so the only way to get the PCR value is to hash the same blocks of memory in the same order.

Platform

Platform DSC libraries for TPM

This is a partial list of libraries that platform DSC file typically uses for TPM.

- `Tpm12DeviceLib|SecurityPkg/Library/Tpm12DeviceLibDTpm/Tpm12DeviceLibDTpm.inf`
- `Tpm2DeviceLib|SecurityPkg/Library/Tpm2DeviceLibTcg2/Tpm2DeviceLibTcg2.inf`
- `Tpm2DeviceLib|SecurityPkg/Library/Tpm2DeviceLibRouter/Tpm2DeviceLibRouterDxe.inf`
- `Tcg2PhysicalPresenceLib|SecurityPkg/Library/PeiTcg2PhysicalPresenceLib/PeiTcg2PhysicalPresenceLib.inf`
- `Tcg2PhysicalPresenceLib|SecurityPkg/Library/DxeTcg2PhysicalPresenceLib/DxeTcg2PhysicalPresenceLib.inf`
- `Tcg2PhysicalPresenceLib|SecurityPkg/Library/SmmTcg2PhysicalPresenceLib/SmmTcg2PhysicalPresenceLib.inf`
- `TpmMeasurementLib|SecurityPkg//Library/DxeTpmMeasurementLib/DxeTpmMeasurementLib.inf`
- `TpmCommLib|SecurityPkg/Library/TpmCommLib/TpmCommLib.inf`
- `Tpm12CommandLib|SecurityPkg/Library/Tpm12CommandLib/Tpm12CommandLib.inf`
- `Tpm2CommandLib|SecurityPkg/Library/Tpm2CommandLib/Tpm2CommandLib.inf`
- `HashLib|SecurityPkg/Library/HashLibBaseCryptoRouter/HashLibBaseCryptoRouterPei.inf`
- `HashLib|SecurityPkg/Library/HashLibBaseCryptoRouter/HashLibBaseCryptoRouterDxe.inf`
- `TcgPpVendorLib|SecurityPkg/Library/TcgPpVendorLibNull/TcgPpVendorLibNull.inf`
- `Tcg2PpVendorLib|SecurityPkg/Library/Tcg2PpVendorLibNull/Tcg2PpVendorLibNull.inf`

Platform DSC drivers for TPM initialization

This is a partial list of drivers that platform DSC file must use to initialize PCI services.

- `SecurityPkg/Tcg/Tcg2Config/Tcg2ConfigPei.inf`
- `SecurityPkg/Tcg/Tcg2ConfigDxe/Tcg2ConfigDxe.inf`
- `SecurityPkg/Tcg/MemoryOverwriteControl/TcgMor.inf`
- `SecurityPkg/Tcg/Tcg2Pei/Tcg2Pei.inf`
- `SecurityPkg/Tcg/Tcg2Dxe/Tcg2Dxe.inf`
- `SecurityPkg/Tcg/Tcg2Smm/Tcg2Smm.inf`
- `SecurityPkg/Tcg/PhysicalPresencePei/PhysicalPresencePei.inf`

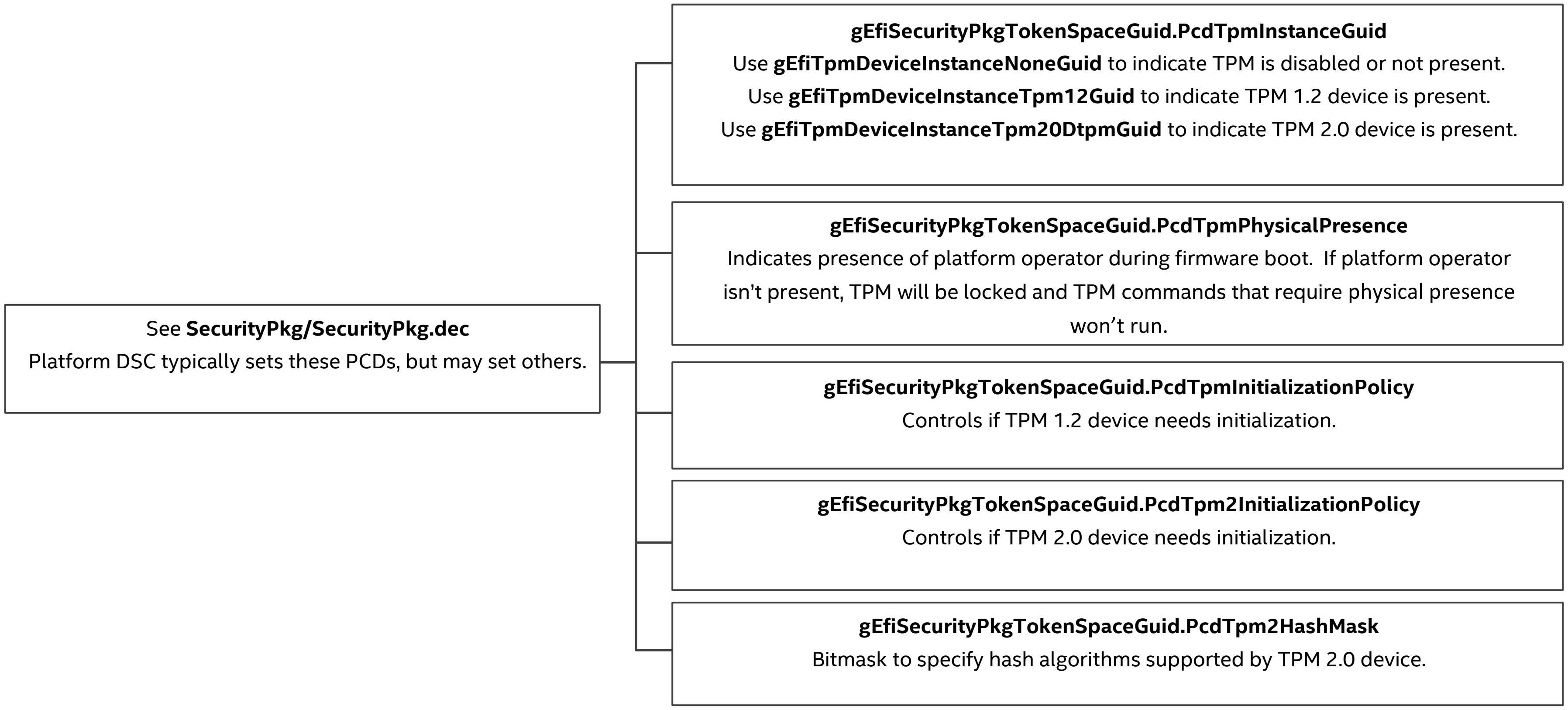
Platform DSC PCDs for TPM initialization

This is a partial list of PCDs that platform DSC file typically sets for TPM.

- `gEfiSecurityPkgTokenSpaceGuid.PcdTpmPhysicalPresence`
- `gEfiSecurityPkgTokenSpaceGuid.PcdTpmInstanceGuid`
  - `Used by code to execute TPM 1.2 or 2.0 commands.`
  - `Set in platform DSC to select TPM type.`
- `gEfiSecurityPkgTokenSpaceGuid.PcdTpmInitializationPolicy`
- `gEfiSecurityPkgTokenSpaceGuid.PcdTpm2InitializationPolicy`
- `gEfiSecurityPkgTokenSpaceGuid.PcdTpm2HashMask`

See **TPM PCDs** for details.

TPM PCDs



See **SecurityPkg/SecurityPkg.dec**
Platform DSC typically sets these PCDs, but may set others.

gEfiSecurityPkgTokenSpaceGuid.PcdTpmInstanceGuid
Use **gEfiTpmDeviceInstanceNoneGuid** to indicate TPM is disabled or not present.
Use **gEfiTpmDeviceInstanceTpm12Guid** to indicate TPM 1.2 device is present.
Use **gEfiTpmDeviceInstanceTpm20DtpmGuid** to indicate TPM 2.0 device is present.

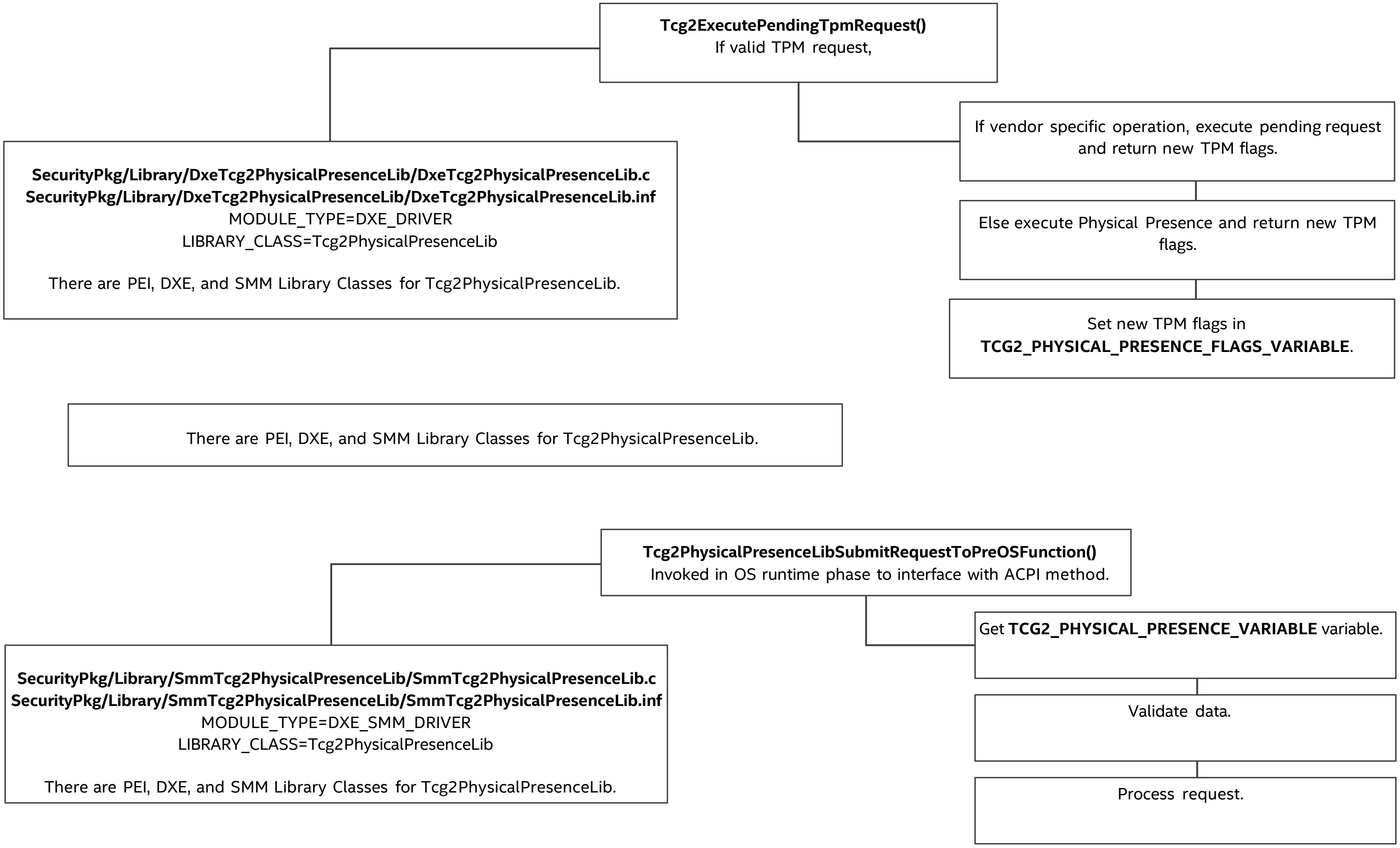
gEfiSecurityPkgTokenSpaceGuid.PcdTpmPhysicalPresence
Indicates presence of platform operator during firmware boot. If platform operator isn't present, TPM will be locked and TPM commands that require physical presence won't run.

gEfiSecurityPkgTokenSpaceGuid.PcdTpmInitializationPolicy
Controls if TPM 1.2 device needs initialization.

gEfiSecurityPkgTokenSpaceGuid.PcdTpm2InitializationPolicy
Controls if TPM 2.0 device needs initialization.

gEfiSecurityPkgTokenSpaceGuid.PcdTpm2HashMask
Bitmask to specify hash algorithms supported by TPM 2.0 device.

Physical Presence



Tcg2ExecutePendingTpmRequest()
If valid TPM request,

SecurityPkg/Library/DxeTcg2PhysicalPresenceLib/DxeTcg2PhysicalPresenceLib.c
SecurityPkg/Library/DxeTcg2PhysicalPresenceLib/DxeTcg2PhysicalPresenceLib.inf
MODULE_TYPE=DXE_DRIVER
LIBRARY_CLASS=Tcg2PhysicalPresenceLib

There are PEI, DXE, and SMM Library Classes for Tcg2PhysicalPresenceLib.

If vendor specific operation, execute pending request and return new TPM flags.

Else execute Physical Presence and return new TPM flags.

Set new TPM flags in **TCG2_PHYSICAL_PRESENCE_FLAGS_VARIABLE**.

There are PEI, DXE, and SMM Library Classes for Tcg2PhysicalPresenceLib.

Tcg2PhysicalPresenceLibSubmitRequestToPreOSFunction()
Invoked in OS runtime phase to interface with ACPI method.

SecurityPkg/Library/SmmTcg2PhysicalPresenceLib/SmmTcg2PhysicalPresenceLib.c
SecurityPkg/Library/SmmTcg2PhysicalPresenceLib/SmmTcg2PhysicalPresenceLib.inf
MODULE_TYPE=DXE_SMM_DRIVER
LIBRARY_CLASS=Tcg2PhysicalPresenceLib

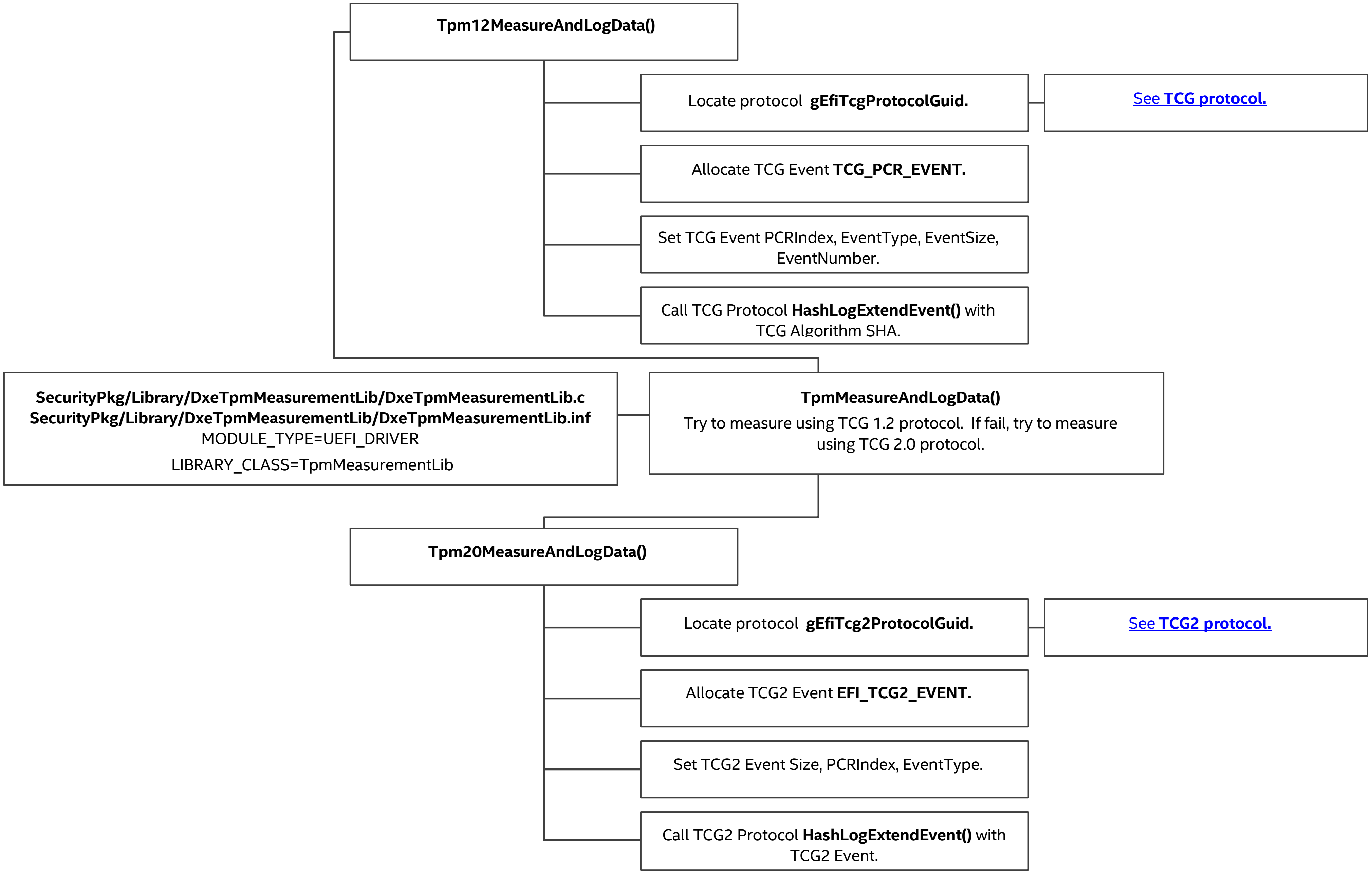
There are PEI, DXE, and SMM Library Classes for Tcg2PhysicalPresenceLib.

Get **TCG2_PHYSICAL_PRESENCE_VARIABLE** variable.

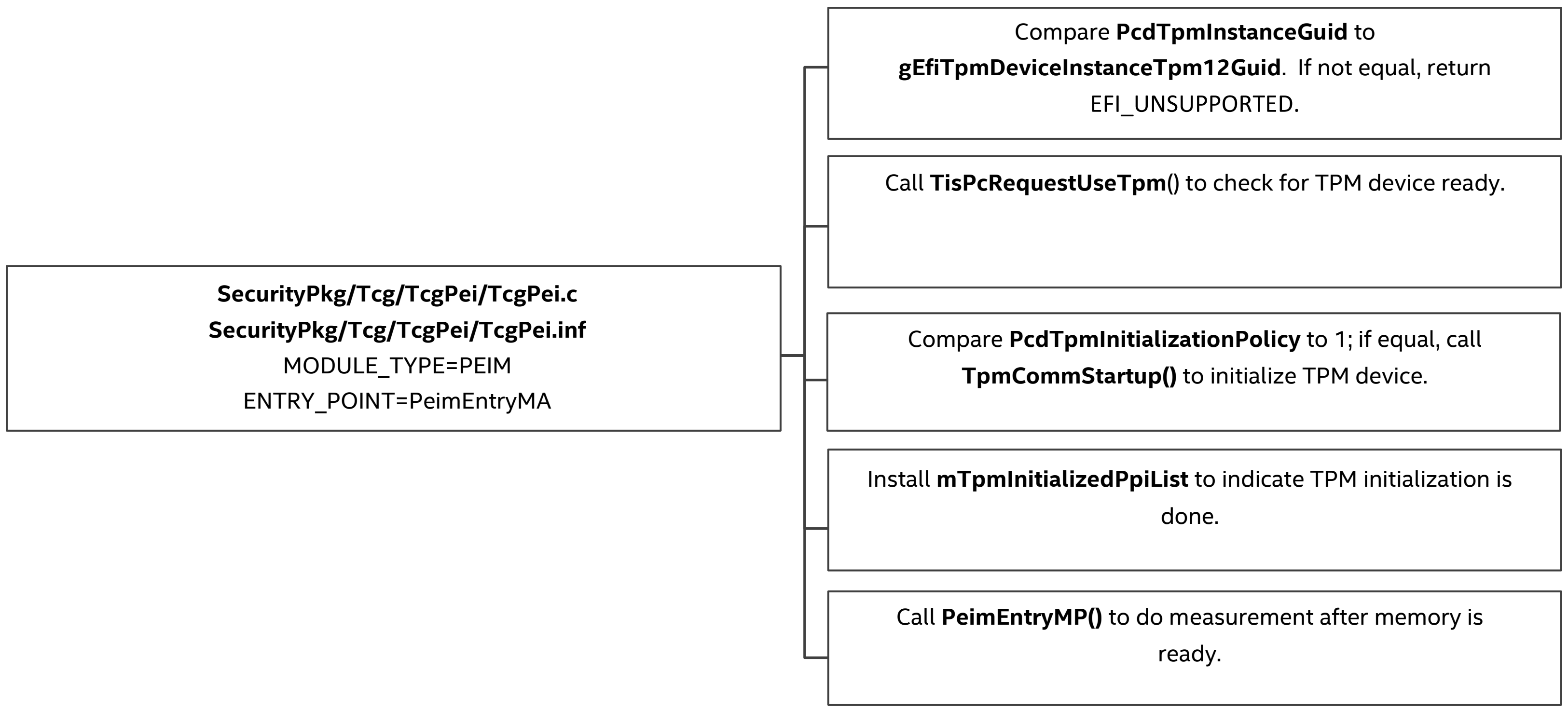
Validate data.

Process request.

Measure


Tpm12MeasureAndLogData()
Locate protocol **gEfiTcgProtocolGuid.**
See **TCG protocol.**
Allocate TCG Event **TCG_PCR_EVENT.**
Set TCG Event PCRIndex, EventType, EventSize, EventNumber.
Call TCG Protocol **HashLogExtendEvent()** with TCG Algorithm SHA.
SecurityPkg/Library/DxeTpmMeasurementLib/DxeTpmMeasurementLib.c
SecurityPkg/Library/DxeTpmMeasurementLib/DxeTpmMeasurementLib.inf
MODULE_TYPE=UEFI_DRIVER
LIBRARY_CLASS=TpmMeasurementLib
TpmMeasureAndLogData()
Try to measure using TCG 1.2 protocol. If fail, try to measure using TCG 2.0 protocol.
Tpm20MeasureAndLogData()
Locate protocol **gEfiTcg2ProtocolGuid.**
See **TCG2 protocol.**
Allocate TCG2 Event **EFI_TCG2_EVENT.**
Set TCG2 Event Size, PCRIndex, EventType.
Call TCG2 Protocol **HashLogExtendEvent()** with TCG2 Event.

TCG (PEI)



SecurityPkg/Tcg/TcgPei/TcgPei.c
SecurityPkg/Tcg/TcgPei/TcgPei.inf
MODULE_TYPE=PEIM
ENTRY_POINT=PeimEntryMA

Compare **PcdTpmInstanceGuid** to **gEfiTpmDeviceInstanceTpm12Guid**. If not equal, return EFI_UNSUPPORTED.

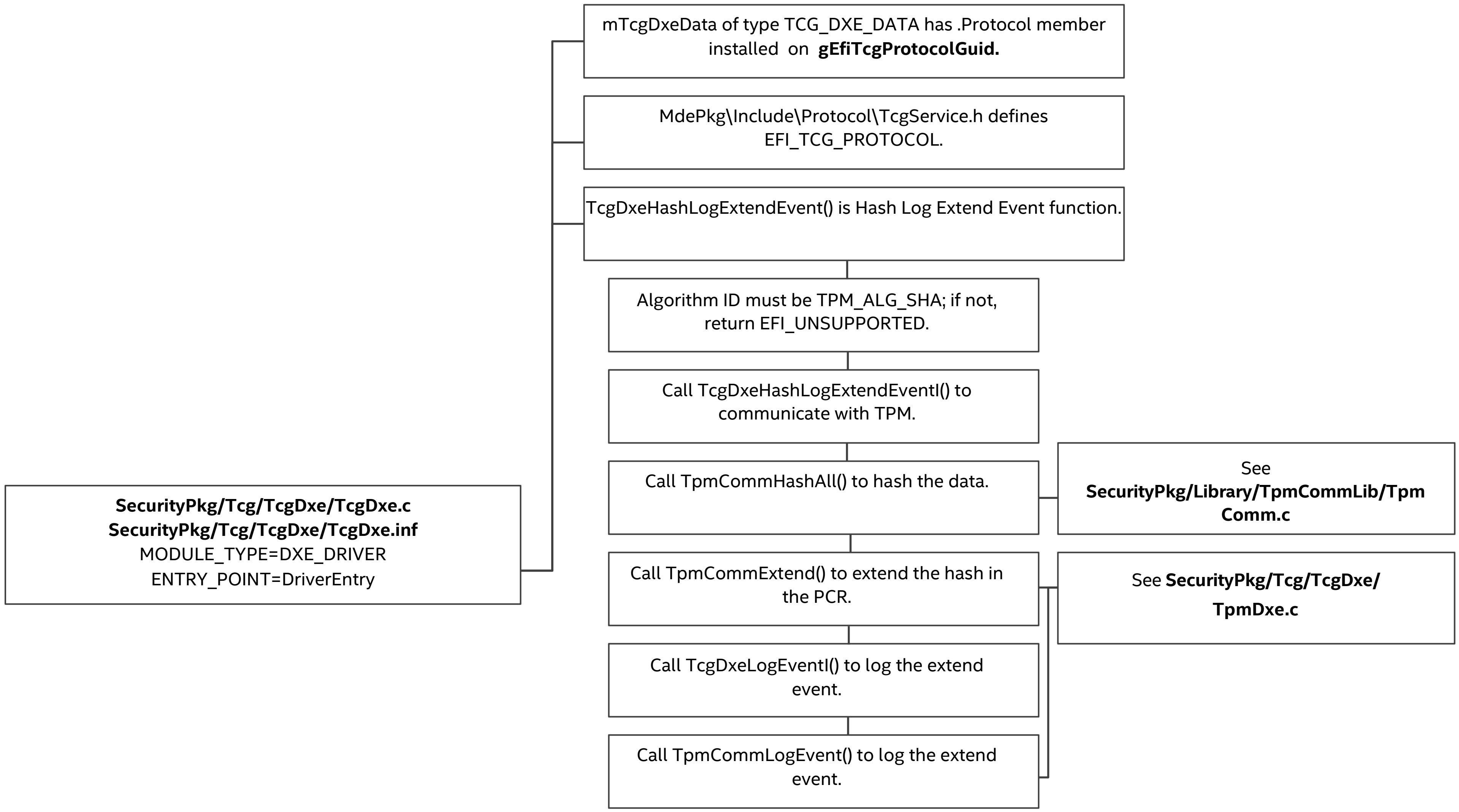
Call **TisPcRequestUseTpm**() to check for TPM device ready.

Compare **PcdTpmInitializationPolicy** to 1; if equal, call **TpmCommStartup()** to initialize TPM device.

Install **mTpmInitializedPpiList** to indicate TPM initialization is done.

Call **PeimEntryMP()** to do measurement after memory is ready.

TCG Protocol (DXE)



SecurityPkg/Tcg/TcgDxe/TcgDxe.c
SecurityPkg/Tcg/TcgDxe/TcgDxe.inf
MODULE_TYPE=DXE_DRIVER
ENTRY_POINT=DriverEntry

mTcgDxeData of type TCG_DXE_DATA has .Protocol member installed on **gEfiTcgProtocolGuid.**

MdePkg\Include\Protocol\TcgService.h defines EFI_TCG_PROTOCOL.

TcgDxeHashLogExtendEvent() is Hash Log Extend Event function.

Algorithm ID must be TPM_ALG_SHA; if not, return EFI_UNSUPPORTED.

Call TcgDxeHashLogExtendEventI() to communicate with TPM.

Call TpmCommHashAll() to hash the data.

See **SecurityPkg/Library/TpmCommLib/TpmComm.c**

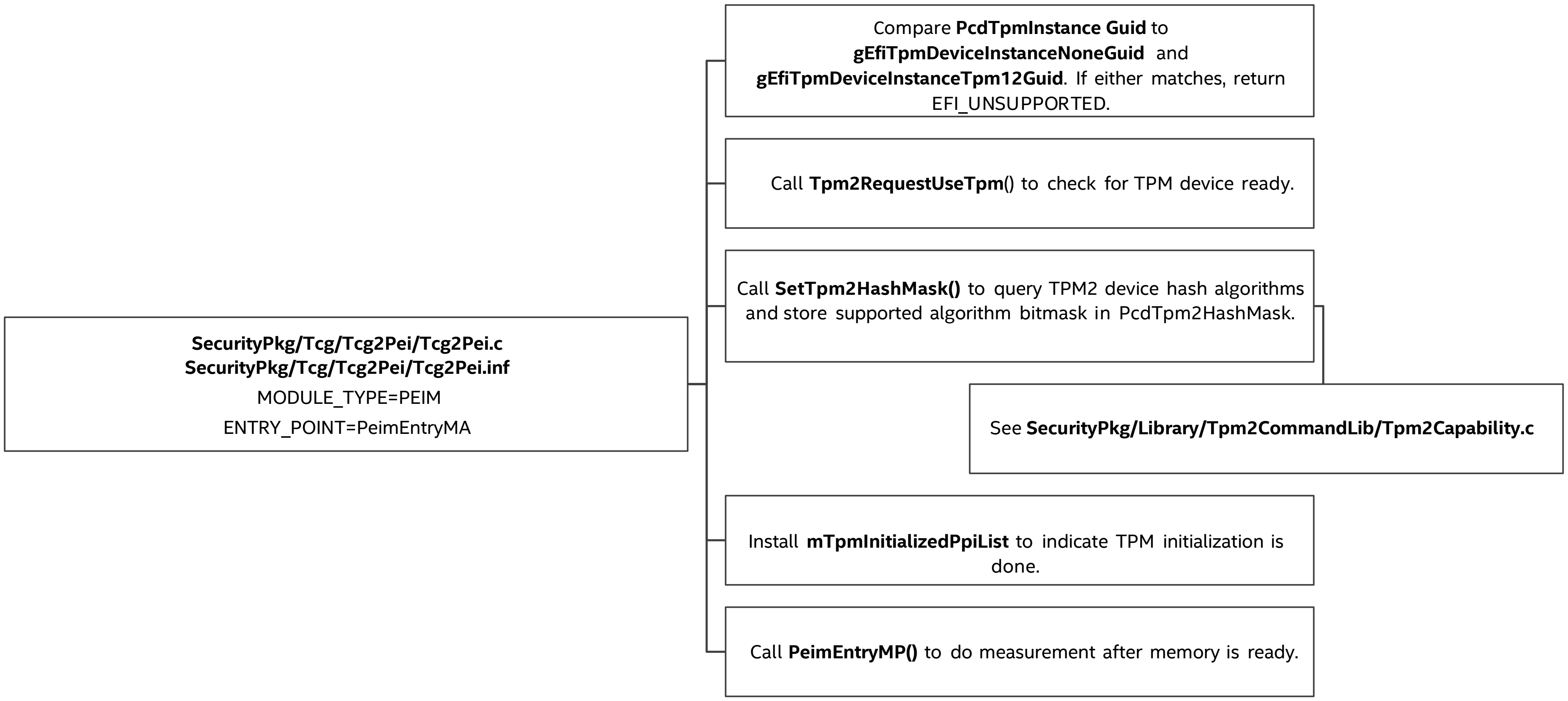
Call TpmCommExtend() to extend the hash in the PCR.

See **SecurityPkg/Tcg/TcgDxe/TpmDxe.c**

Call TcgDxeLogEventI() to log the extend event.

Call TpmCommLogEvent() to log the extend event.

TCG2 (PEI)



SecurityPkg/Tcg/Tcg2Pei/Tcg2Pei.c
SecurityPkg/Tcg/Tcg2Pei/Tcg2Pei.inf
MODULE_TYPE=PEIM
ENTRY_POINT=PeimEntryMA

Compare **PcdTpmInstance Guid** to **gEfiTpmDeviceInstanceNoneGuid** and **gEfiTpmDeviceInstanceTpm12Guid**. If either matches, return EFI_UNSUPPORTED.

Call **Tpm2RequestUseTpm**() to check for TPM device ready.

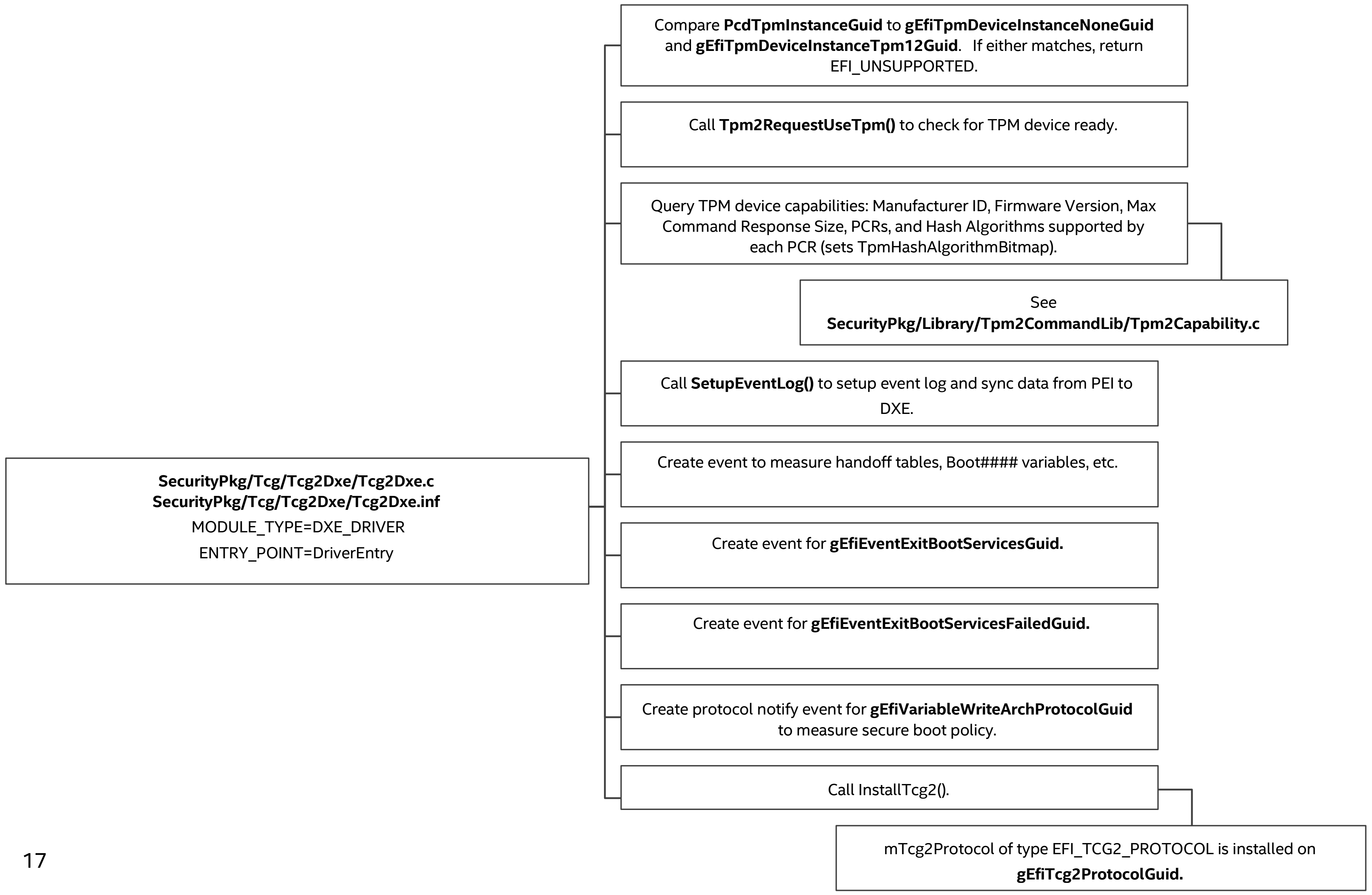
Call **SetTpm2HashMask()** to query TPM2 device hash algorithms and store supported algorithm bitmask in PcdTpm2HashMask.

See **SecurityPkg/Library/Tpm2CommandLib/Tpm2Capability.c**

Install **mTpmInitializedPpiList** to indicate TPM initialization is done.

Call **PeimEntryMP()** to do measurement after memory is ready.

TCG2 Protocol (DXE)



SecurityPkg/Tcg/Tcg2Dxe/Tcg2Dxe.c
SecurityPkg/Tcg/Tcg2Dxe/Tcg2Dxe.inf
MODULE_TYPE=DXE_DRIVER
ENTRY_POINT=DriverEntry

Compare **PcdTpmInstanceGuid** to **gEfiTpmDeviceInstanceNoneGuid** and **gEfiTpmDeviceInstanceTpm12Guid**. If either matches, return EFI_UNSUPPORTED.

Call **Tpm2RequestUseTpm()** to check for TPM device ready.

Query TPM device capabilities: Manufacturer ID, Firmware Version, Max Command Response Size, PCRs, and Hash Algorithms supported by each PCR (sets TpmHashAlgorithmBitmap).

See **SecurityPkg/Library/Tpm2CommandLib/Tpm2Capability.c**

Call **SetupEventLog()** to setup event log and sync data from PEI to DXE.

Create event to measure handoff tables, Boot#### variables, etc.

Create event for **gEfiEventExitBootServicesGuid.**

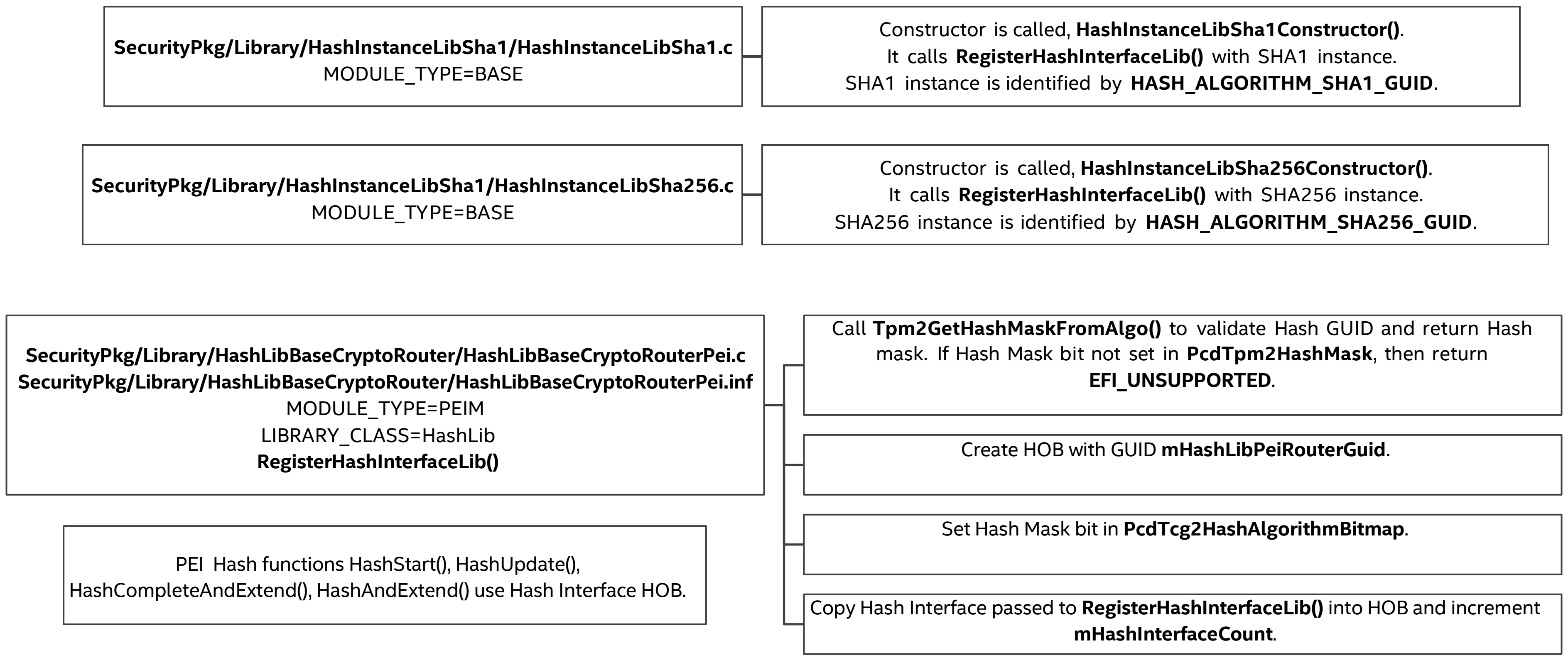
Create event for **gEfiEventExitBootServicesFailedGuid.**

Create protocol notify event for **gEfiVariableWriteArchProtocolGuid** to measure secure boot policy.

Call InstallTcg2().

mTcg2Protocol of type EFI_TCG2_PROTOCOL is installed on **gEfiTcg2ProtocolGuid.**

Hash



SecurityPkg/Library/HashInstanceLibSha1/HashInstanceLibSha1.c
MODULE_TYPE=BASE

Constructor is called, **HashInstanceLibSha1Constructor()**.
It calls **RegisterHashInterfaceLib()** with SHA1 instance.
SHA1 instance is identified by **HASH_ALGORITHM_SHA1_GUID**.

SecurityPkg/Library/HashInstanceLibSha1/HashInstanceLibSha256.c
MODULE_TYPE=BASE

Constructor is called, **HashInstanceLibSha256Constructor()**.
It calls **RegisterHashInterfaceLib()** with SHA256 instance.
SHA256 instance is identified by **HASH_ALGORITHM_SHA256_GUID**.

SecurityPkg/Library/HashLibBaseCryptoRouter/HashLibBaseCryptoRouterPei.c
SecurityPkg/Library/HashLibBaseCryptoRouter/HashLibBaseCryptoRouterPei.inf
MODULE_TYPE=PEIM
LIBRARY_CLASS=HashLib
RegisterHashInterfaceLib()

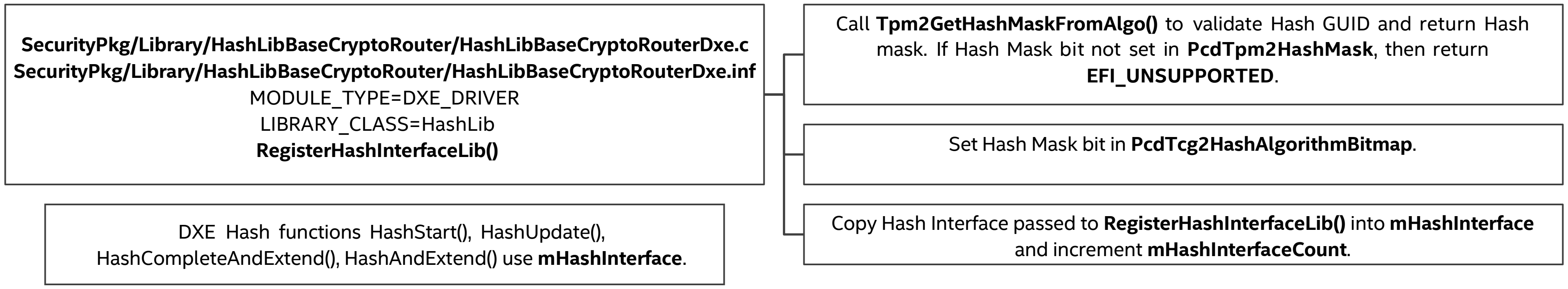
Call **Tpm2GetHashMaskFromAlgo()** to validate Hash GUID and return Hash mask. If Hash Mask bit not set in **PcdTpm2HashMask**, then return **EFI_UNSUPPORTED**.

Create HOB with GUID **mHashLibPeiRouterGuid**.

Set Hash Mask bit in **PcdTcg2HashAlgorithmBitmap**.

Copy Hash Interface passed to **RegisterHashInterfaceLib()** into HOB and increment **mHashInterfaceCount**.

PEI Hash functions HashStart(), HashUpdate(), HashCompleteAndExtend(), HashAndExtend() use Hash Interface HOB.



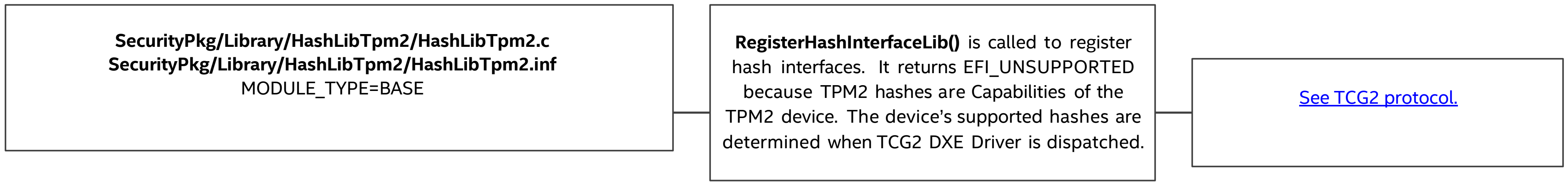
SecurityPkg/Library/HashLibBaseCryptoRouter/HashLibBaseCryptoRouterDxe.c
SecurityPkg/Library/HashLibBaseCryptoRouter/HashLibBaseCryptoRouterDxe.inf
MODULE_TYPE=DXE_DRIVER
LIBRARY_CLASS=HashLib
RegisterHashInterfaceLib()

DXE Hash functions HashStart(), HashUpdate(), HashCompleteAndExtend(), HashAndExtend() use **mHashInterface**.

Call **Tpm2GetHashMaskFromAlgo()** to validate Hash GUID and return Hash mask. If Hash Mask bit not set in **PcdTpm2HashMask**, then return **EFI_UNSUPPORTED**.

Set Hash Mask bit in **PcdTcg2HashAlgorithmBitmap**.

Copy Hash Interface passed to **RegisterHashInterfaceLib()** into **mHashInterface** and increment **mHashInterfaceCount**.



SecurityPkg/Library/HashLibTpm2/HashLibTpm2.c
SecurityPkg/Library/HashLibTpm2/HashLibTpm2.inf
MODULE_TYPE=BASE

RegisterHashInterfaceLib() is called to register hash interfaces. It returns EFI_UNSUPPORTED because TPM2 hashes are Capabilities of the TPM2 device. The device's supported hashes are determined when TCG2 DXE Driver is dispatched.

See TCG2 protocol.

References

All EDK II core package code referenced in this document is located in the GitHub EDK II repository: https://github.com/tianocore/edk2

For more in-depth information about EDK II, visit the Intel® Firmware: Beyond BIOS page: https://firmware.intel.com/blog/beyond-bios

Visit TianoCore.org for more EDK II documentation and EDK II projects: https://github.com/tianocore/tianocore.github.io/wiki/EDK-II-Documents

Authors

Hamel, Lee M Hamel: lee.m.hamel@intel.com
Stephen C Bekefi: stephen.c.bekefi@intel.com