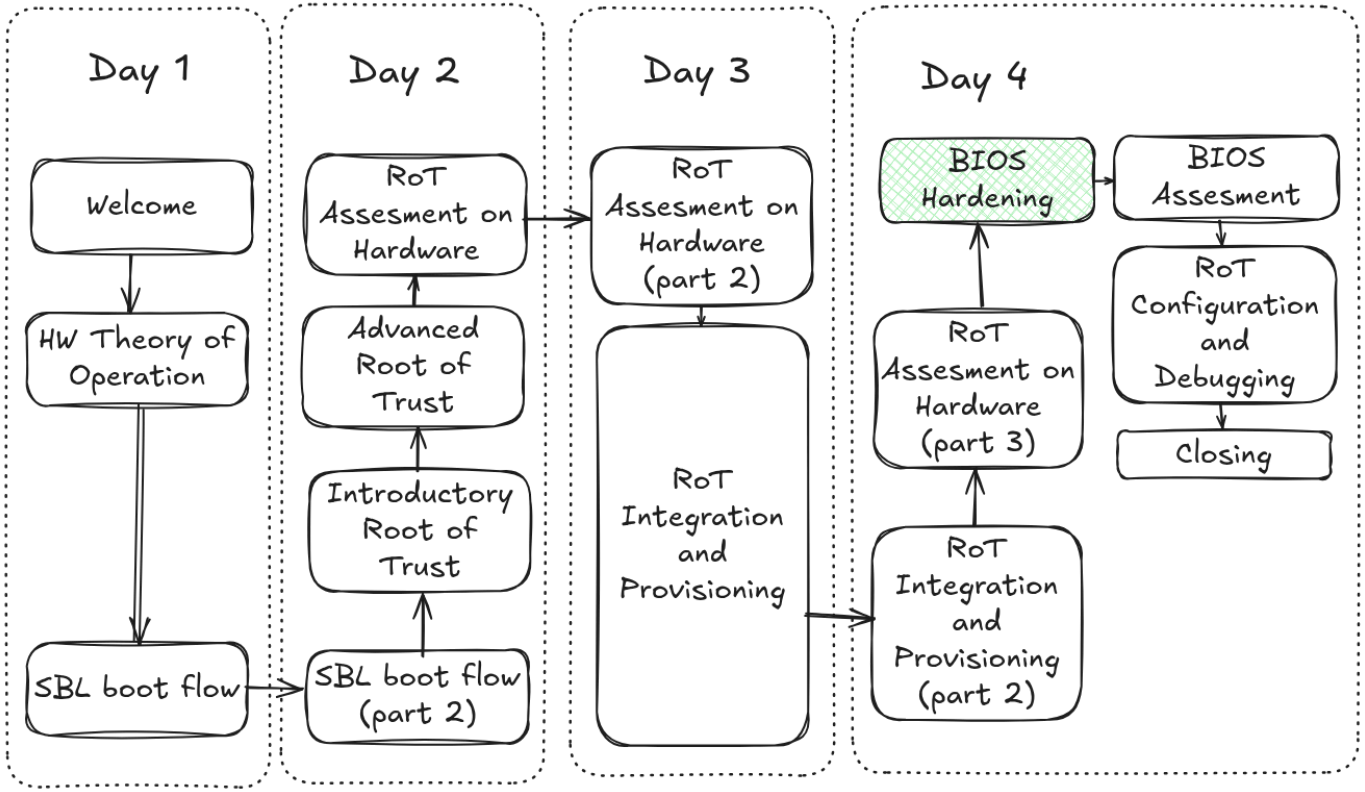
DS09SBL: Dasharo TrustRoot Training

BIOS Hardening

Dasharo TrustRoot Training

Where we are in the course



Goals of the Presentation

In this presentation, we aim to:

- Understand basics of chipset-based SPI flash protection and security.
- Dive deeply in Slim Bootloader Verified Boot.
- Dive deeply in Slim Bootloader Measured Boot.

Requirements

- Basic understanding of SPI protocol and SPI flash theory of operation.
- Basic understanding of Slim Bootloader boot flow.
- Basic understanding of TPM.

SPI security features

Goal of this lecture

The subject of this lecture is to describe the SPI flash protection mechanisms from the chipset and SPI flash perspective. The subjects covered:

- SPI flash overview
- SPI flash protections (PCH/chipset)
- SPI flash protections (SPI chip)

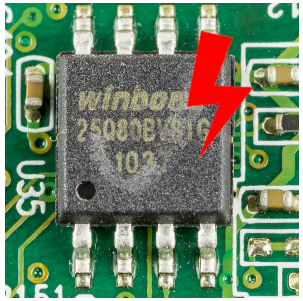
SPI flash photo © Raimond Spekking / CC BY-SA 4.0 (via Wikimedia Commons)

Derivative work notice

Derived from Xeno Kovah's 'Architecture 4001: x86-64 Intel Firmware Attack & Defense' class, available at https://ost2.fyi, which itself was derived from John Butterworth & Xeno Kovah's 'Advanced Intel x86: BIOS and SMM' class posted at http://opensecuritytraining.info/IntroBIOS.html

SPI flash overview

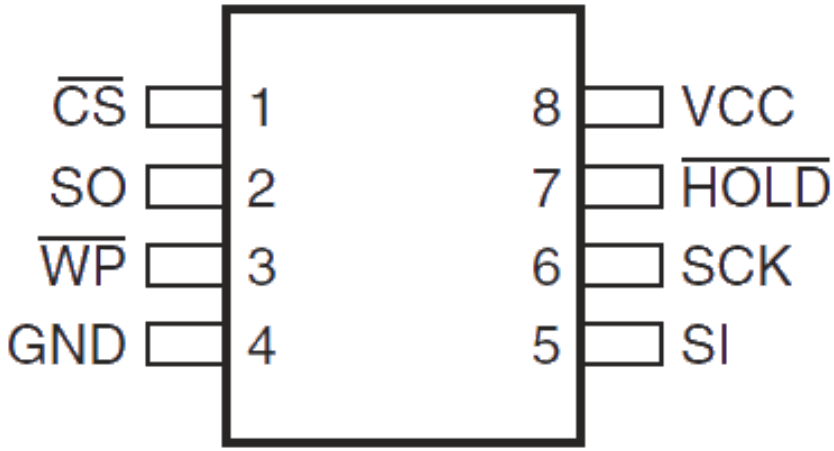
SPI flash threats



- One of the primary goals of an adversary who decides to compromise a BIOS is to achieve stealthy persistence
  - Because very few people know how to detect a BIOS-resident adversary
- To achieve persistence, the adversary will have to figure out a way to write to the BIOS flash

SPI flash photo © Raimond Spekking / CC BY-SA 4.0 (via Wikimedia Commons)

Serial Peripheral Interface (SPI)



- Intel's ICH/PCH implements an SPI interface for the BIOS flash device
- SPI is not just for the BIOS, but also used by the Management Engine (ME), Gigabit Ethernet (GbE), and others
- Modern components can be up to 2Gb in density using 1.8V and 3V with speed up to 133MHz
- Typical SPI flash devices can be up to 16 MB
- SPI controller can support multiple devices for bigger maximum addressable space

SPI overview

- Intel defines a set of minimum requirements for a chip to support in SPI Programming Guides in ME toolkits
  - Chips will generally support more than just that bare minimum
- We'll be covering Intel's implementation and interface to SPI, not really the SPI protocol itself
- For register references we will be using Intel 300 Series chipset PCH datasheet vol.1 and Intel 300 Series chipset PCH datasheet vol.2 unless stated otherwise

SPI flash protection (PCH/chipset)

BIOS boot straps (BBS)

35.1.28 General Control And Status (GCS)—Offset 274Ch

Access Method

Type: MSG Register
(Size: 32 bits)

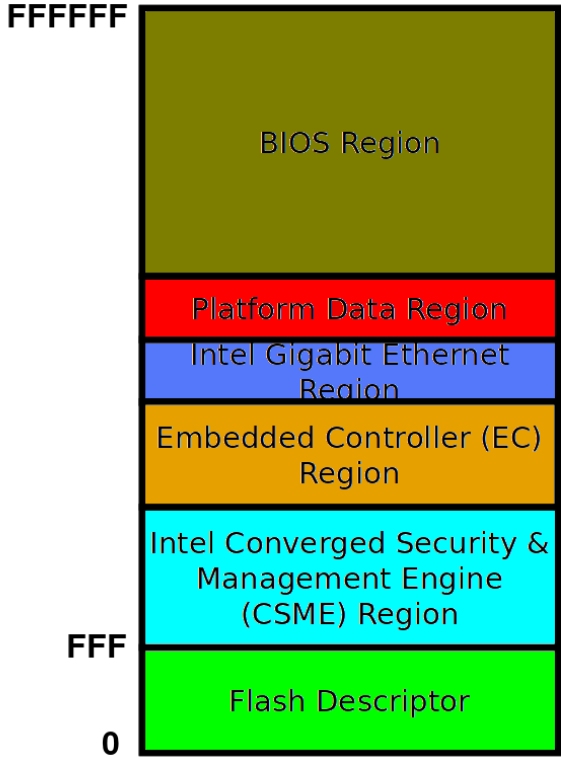
Device:
Function:

Default: 0h

10	0h RW/V/L	**Boot BIOS Strap (BBS):** This field determines the destination of accesses to the BIOS memory range. Bits Description '0b': SPI '1b': LPC/eSPI When SPI or LPC/eSPI is selected, the range that is decoded is further qualified by other configuration bits described in the respective sections. The value in this field can be overwritten by software as long as the BIOS Interface Lock-Down (bit 0) and MPC.SRL are not set. Register Attribute: Static. This register is Read-Only if MPC.SRL field is set
9:1	0h RO	Reserved.
0	0h RW/O	**BIOS Interface Lock-Down (BILD):** When set, prevents GCS.BBS from being changed. This bit can only be written from 0 to 1 once. Register Attribute: Static.

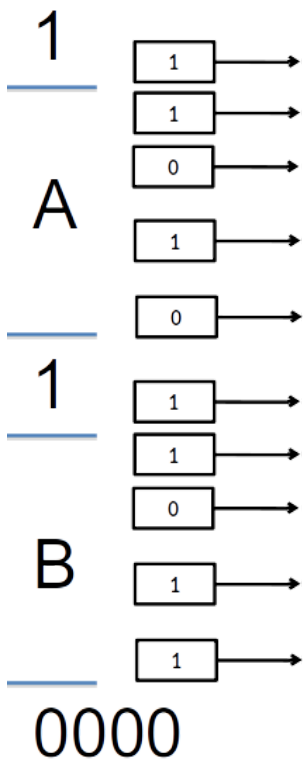
Flash Descriptor (FD)

FD - Flash regions



- Each register with permissions (for each bus master) have the same layout
- These registers are located at Flash Masters Base Address (FMBA) inside FD and are called FLMSTRn where `n` is the ID of bus master:
  - FLMSTR1 (CPU/BIOS)
  - FLMSTR2 (ME)
  - FLMSTR3 (GbE)
- Detailed descriptions with register addresses may be found in SPI Programming Guide in ME toolkits, e.g. for 300 series PCH:

Region Name	Starting Address
Signature	10h
Component FCBA	30h
Regions FRBA	40h
Masters FMBA	80h
PCH Straps FPSBA	100h
MDTBA	C00h
PMC Straps	C14h
CPU Straps	C2Ch
Intel® ME Straps	C3Ch
Register Init FIBA	340h



Bits	Description
31:29	Reserved, must be zero
28	**Platform Data Region Write Access**. If the bit is set, this master can erase and write that particular region through register accesses.
27	**GbE Region Write Access**. If the bit is set, this master can erase and write that particular region through register accesses.
26	**ME Region Write Access**. If the bit is set, this master can erase and write that particular region through register accesses.
25	**Host CPU/BIOS Master Region Write Access**. If the bit is set, this master can erase and write that particular region through register accesses. Bit 25 is a don't care as the primary master always has read/write permissions to it's primary region
24	**Flash Descriptor Region Write Access**. If the bit is set, this master can erase and write that particular region through register accesses.
23:21	Reserved, must be zero
20	**Platform Data Region Read Access**. If the bit is set, this master can read that particular region through register accesses.
19	**GbE Region Read Access**. If the bit is set, this master can read that particular region through register accesses.
18	**ME Region Read Access**. If the bit is set, this master can read that particular region through register accesses.
17	**Host CPU/BIOS Master Region Read Access**. If the bit is set, this master can read that particular region through register accesses. Bit 17 is a don't care as the primary master always has read/write permissions to it's primary region
16	**Flash Descriptor Region Read Access**. If the bit is set, this master can read that particular region through register accesses.
15:0	**Requester ID**. This is the Requester ID of the Host processor. This must be set to 0000h.

FD - Permissions bypass

Flash descriptor bus mater permissions may be bypassed in two ways:

- **HECI HMRFPO_ENABLE command** - Host ME Region Flash Protection Override Enable
  - Can be sent only by BIOS in a specified timeframe during boot
  - Requires global reset to apply
  - Tells ME not to enforce flash descriptor bus mater permissions
  - **Mitigation: BIOS must send HMRFPO_LOCK command to prevent HMRFPO_ENABLE to succeed**
- **Flash Descriptor Security Override Pin Strap**
  - When asserted (pulled up to high voltage) during system power on, ME will not enforce flash descriptor bus mater permissions
  - This pin corresponds to HDA_SDO (Intel HD Audio signal on PCH)
  - Some boards expose this pin on a pin header and allow overriding the flash descriptor bus master permissions
  - **Mitigation: ensure this pin is not exposed on any pin header on your board**

PCH SPI flash access

Direct Access

- This applies to MMIO accesses (mapped to high-memory: [4GB; 4GB - flash size])
- **SPI bus-masters (e.g. CPU vs. ME vs. GbE) are allowed to read only the regions that were specified in the flash descriptor Master section**
- There is no possibility to issue SPI write cycles with direct access

Register Access

- Access a region by programming the SPI controller registers for address to access, type of access, size of access, etc.
- **SPI bus-masters (e.g. CPU vs. ME vs. GbE) are allowed to read/write only the regions that were specified in flash descriptor Master section**
- The only way to issue write/erase cycles to the SPI flash from the host
- Refer to OST SPI FLash programming for details on how to use Register Access method

8.1.8 BIOS Control (BIOS_SPI_BC)—Offset DCh

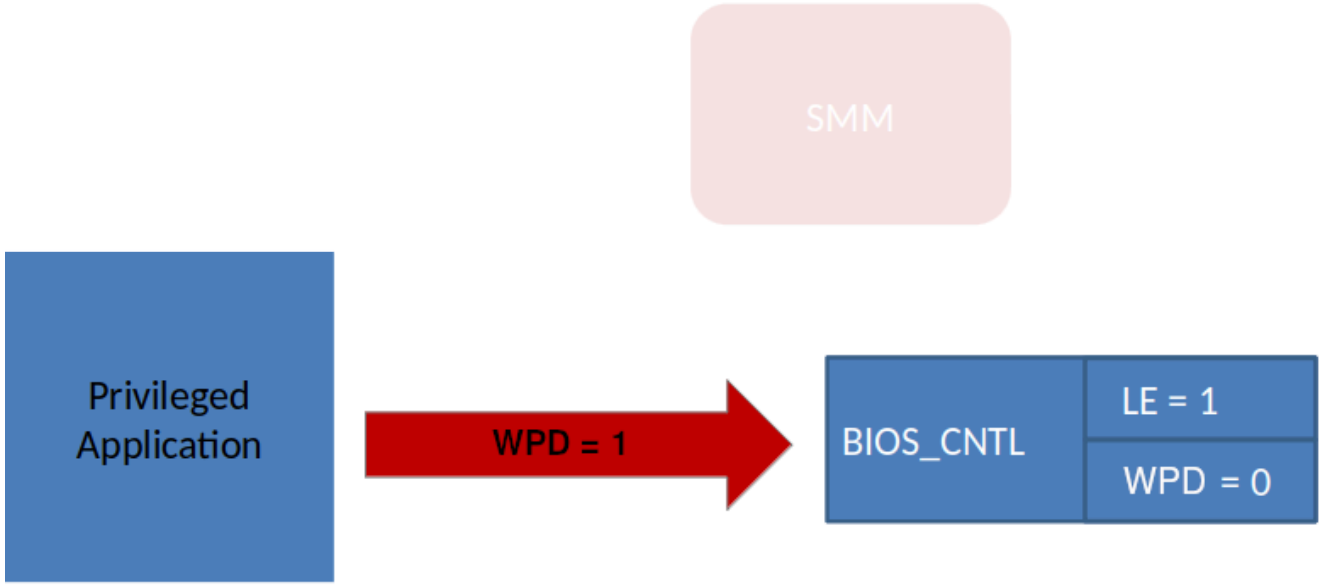
Access Method

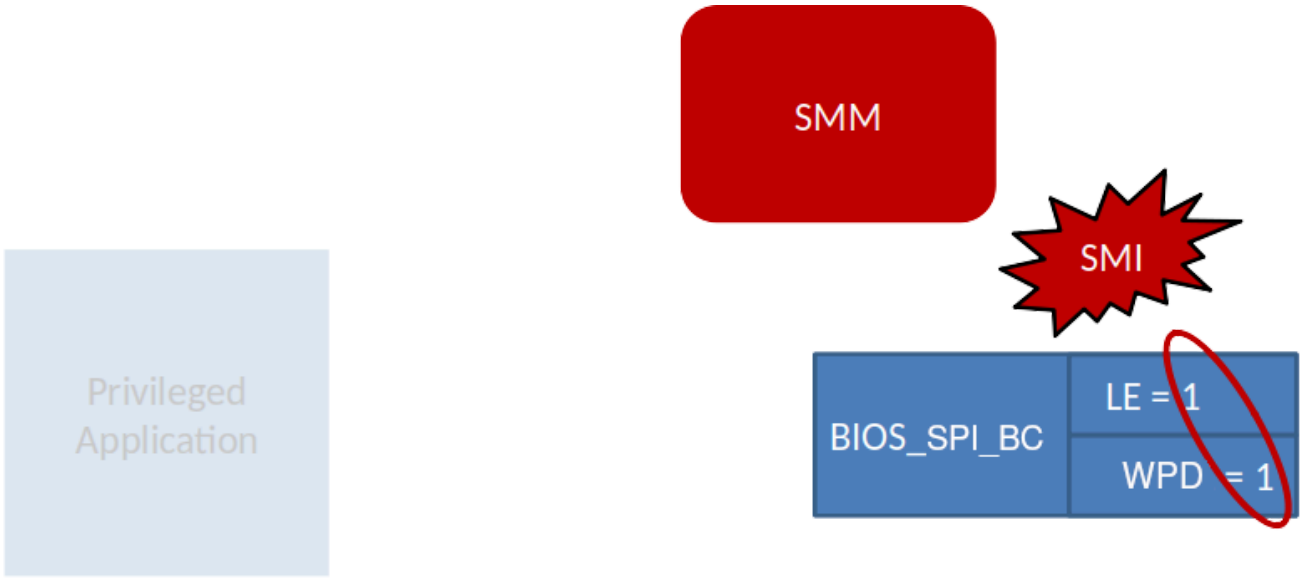
Type: CFG Register
(Size: 32 bits)

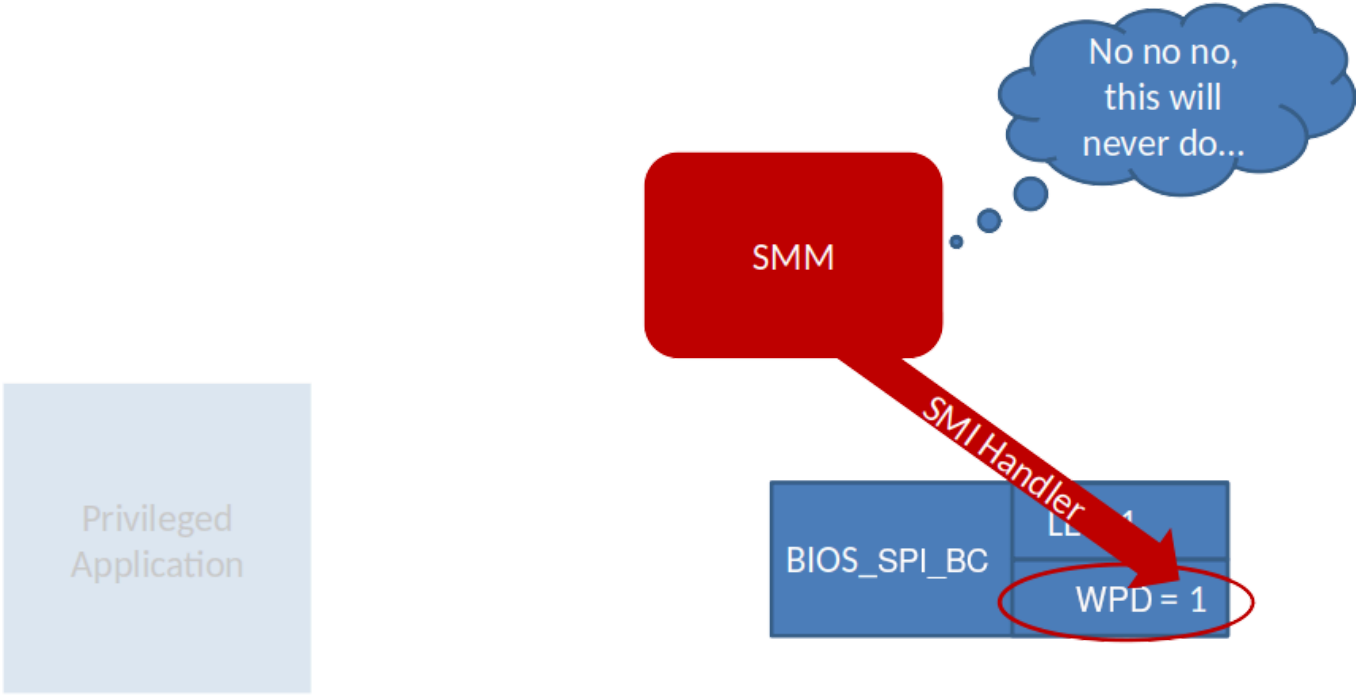
Device: 31
Function: 5

Default: 28h

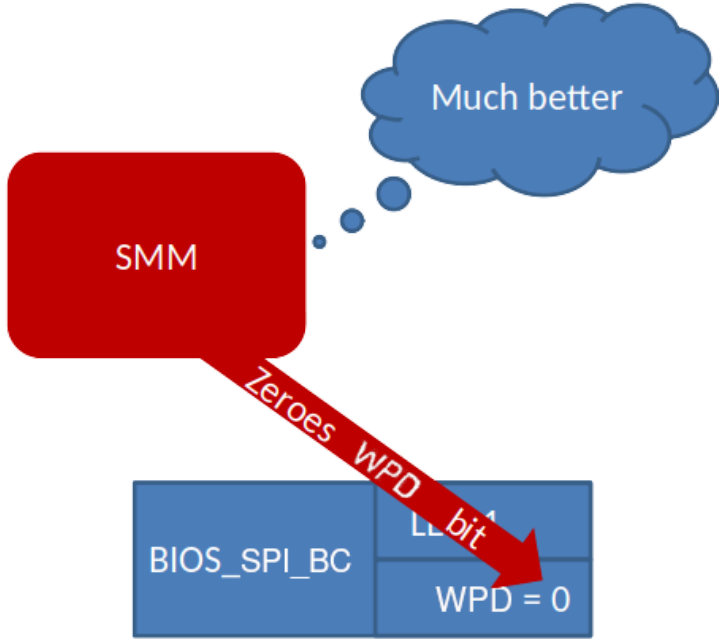
Bit Range	Default and Access	Field Name (ID): Description
5	1h RW/L	**Enable InSMM.STS (EISS):** When this bit is set, the BIOS region is not writable until the CPU sets the InSMM.STS bit. If this bit [5] is set, then WPD must be a '1' and InSMM.STS(0xFED3_0880[0]) must be '1' also in order to write to BIOS region of SPI Flash. If this bit [5] is clear, then the InSMM.STS is a do not care. This bit is locked by LE
4	0h RO/V	**Top Swap Status (TSS):** This bit provides a read-only path to view the state of the Top Swap bit. It is duplicated here to be consistent with the LPC version of the BC register.
3:2	2h RW	**SPI Read Configuration (SRC):** These bits are located in PCI Config space to allow them to be set early in the boot flow. This 2-bit field controls two policies related to BIOS reads on the SPI interface: Bit 3- Prefetch Enable Bit 2- Cache Disable Settings are summarized below: 00 = No prefetching, but caching enabled. Direct Memory reads load the read buffer cache with valid data, allowing repeated reads to the same range to complete quickly 01 = No prefetching and no caching. One-to-one correspondence of host BIOS reads to SPI cycles. This value can be used to invalidate the cache. 10 = Prefetching and Caching enabled. This mode is used for long sequences of short reads to consecutive addresses (i.e. shadowing) 11 = Illegal. Caching must be enabled when Prefetching is enabled.
1	0h RW/L	**Lock Enable (LE):** When set, setting the WPD bit will cause SMI. When cleared, setting the WPD bit will not cause SMI. Once set, this bit can only be cleared by a PLTRST#. When this bit is set, EISS - bit [5] of this register is locked down.
0	0h RW	**Write Protect Disable (WPD):** When set, access to the BIOS space is enabled for both read and write cycles to BIOS. When cleared, only read cycles are permitted to the FWH or SPI flash. When this bit is written from a '0' to a '1' and the LE bit is also set, an SMI# is generated. This ensures that only SMM code can update BIOS.

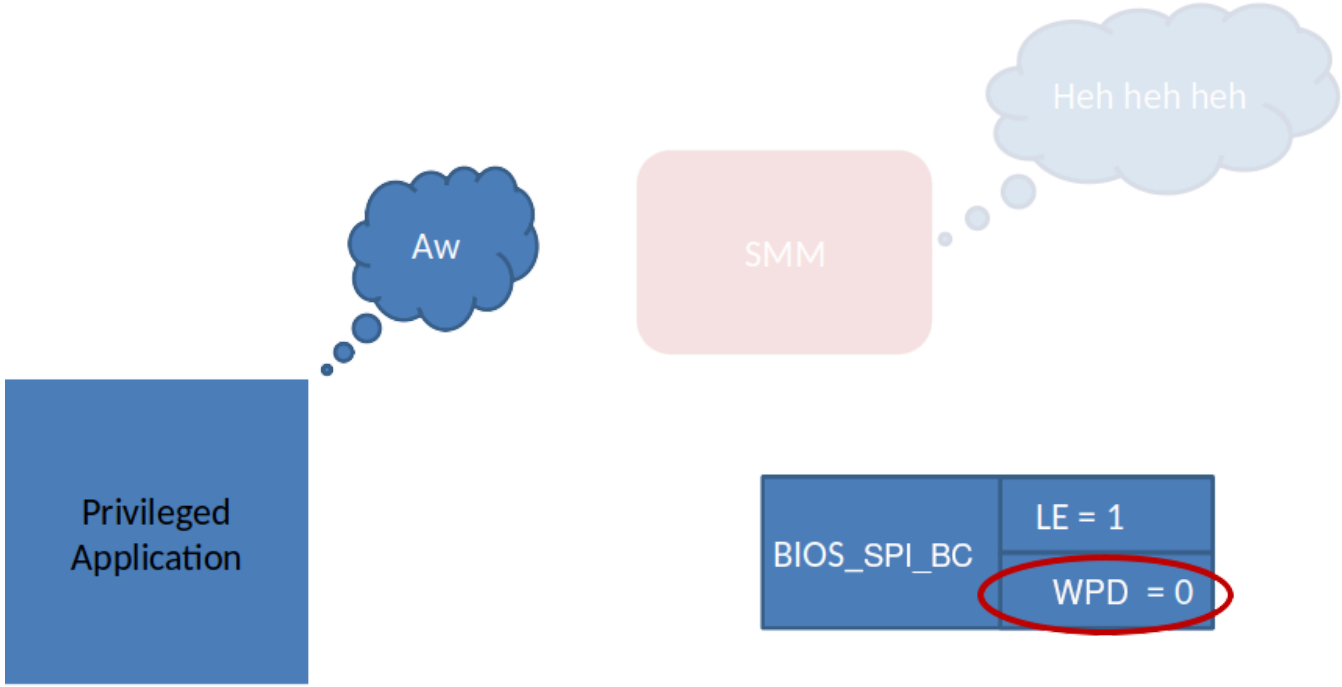






Privileged Application





8.1.8 BIOS Control (BIOS_SPI_BC)—Offset DCh

Access Method

Type: CFG Register
(Size: 32 bits)

Device: 31
Function: 5

Default: 28h

Bit Range	Default and Access	Field Name (ID): Description
5	1h RW/L	**Enable InSMM.STS (EISS):** When this bit is set, the BIOS region is not writable until the CPU sets the InSMM.STS bit. If this bit [5] is set, then WPD must be a '1' and InSMM.STS(0xFED3_0880[0]) must be '1' also in order to write to BIOS region of SPI Flash. If this bit [5] is clear, then the InSMM.STS is a do not care. This bit is locked by LE
4	0h RO/V	**Top Swap Status (TSS):** This bit provides a read-only path to view the state of the Top Swap bit. It is duplicated here to be consistent with the LPC version of the BC register.
3:2	2h RW	**SPI Read Configuration (SRC):** These bits are located in PCI Config space to allow them to be set early in the boot flow. This 2-bit field controls two policies related to BIOS reads on the SPI interface: Bit 3- Prefetch Enable Bit 2- Cache Disable Settings are summarized below: 00 = No prefetching, but caching enabled. Direct Memory reads load the read buffer cache with valid data, allowing repeated reads to the same range to complete quickly 01 = No prefetching and no caching. One-to-one correspondence of host BIOS reads to SPI cycles. This value can be used to invalidate the cache. 10 = Prefetching and Caching enabled. This mode is used for long sequences of short reads to consecutive addresses (i.e. shadowing) 11 = Illegal. Caching must be enabled when Prefetching is enabled.
1	0h RW/L	**Lock Enable (LE):** When set, setting the WPD bit will cause SMI. When cleared, setting the WPD bit will not cause SMI. Once set, this bit can only be cleared by a PLTRST#. When this bit is set, EISS - bit [5] of this register is locked down.
0	0h RW	**Write Protect Disable (WPD):** When set, access to the BIOS space is enabled for both read and write cycles to BIOS. When cleared, only read cycles are permitted to the FWH or SPI flash. When this bit is written from a '0' to a '1' and the LE bit is also set, an SMI# is generated. This ensures that only SMM code can update BIOS.

Table 27-4. Causes of SMI and SCI (Sheet 1 of 2)

Cause	SCI	SMI	Additional Enables (Note 1)	Where Reported
PME#	Yes	Yes	PME_EN=1	PME_STS
PME_B0 (Internal, Bus 0, PME-Capable Agents)	Yes	Yes	PME_B0_EN=1	PME_B0_STS
PCI Express* PME Messages	Yes	Yes	PCI_EXP_EN=1 (Not enabled for SMI)	PCI_EXP_STS
PCI Express Hot-Plug Message	Yes	Yes	HOT_PLUG_EN=1 (Not enabled for SMI)	HOT_PLUG_STS
Power Button Press	Yes	Yes	PWRBTN_EN=1	PWRBTN_STS
Power Button Override (Note 6)	Yes	No	None	PWRBTNOR_STS
RTC Alarm	Yes	Yes	RTC_EN=1	RTC_STS
ACPI Timer overflow (2.34 seconds)	Yes	Yes	TMROF_EN=1	TMROF_STS
GPIO (Note 8)	Yes	Yes		
LAN_WAKE#	Yes	Yes	SCI_EN=0, LAN_WAKE_EN=1	LAN_WAKE_STS
TCO SCI message from processor	Yes	No	None	CPUSCI_STS
TCO SCI Logic	Yes	No	TCOSCI_EN=1	TCOSCI_STS
TCO SMI Logic	No	Yes	TCO_EN=1	TCO_STS
TCO SMI – Year 2000 Rollover	No	Yes	None	NEWCENTURY_STS
TCO SMI – TCO TIMEROUT	No	Yes	None	TIMEOUT
TCO SMI – OS writes to TCO_DAT_IN register	No	Yes	None	OS_TCO_SMI
TCO SMI – NMI occurred (and NMIs mapped to SMI)	No	Yes	NMI2SMI_EN=1	TCO_STS, NMI2SMI_STS
TCO SMI – INTRUDER# signal goes active	No	Yes	INTRD_SEL=10	INTRD_DET
TCO SMI – Changes of the WPD (Write Protect Disable) bit from 0 to 1	No	Yes	LE (Lock Enable)=1	BIOSWR_STS
TCO SMI – Write attempted to BIOS	No	Yes	WPD=0	BIOSWR_STS
BIOS_RLS written to 1 (Note 7)	Yes	No	GBL_EN=1	GBL_STS
GBL_RLS written to	No	Yes	BIOS_EN=1	BIOS_STS

5.2.4 SMI Control and Enable (SMI_EN)—Offset 30h

Lockable: No
Usage: ACPI or Legacy
Power Well: Primary
Note: This register is symmetrical to the SMI Status Register.

Access Method

Type:IO Register (Size: 32 bits)	**Device:** **Function:**

Default:2h

Bit Range	Default and Access	Field Name (ID): Description
26:18	0h RO	Reserved.
17	0h RW	**Legacy USB 2 SMI# Enable (LEGACY_USB2_EN):** Enables legacy USB2 logic to cause SMI#.
16:15	0h RO	Reserved.
14	0h RW	**Periodic Enable (PERIODIC_EN):** Setting this bit will cause the Intel PCH to generate an SMI# when the PERIODIC_STS bit is set in the SMI_STS register. This bit is reset by PLTRST# assertion.
13	0h RW/L	**TCO Enable (TCO_EN):** 1 = Enables the TCO logic to generate SMI#. 0 = Disables TCO logic from generating an SMI#. If the NMI2SMI_EN bit is set, then SMI's that are caused by NMI's (i.e. rerouted) will not be gated by the TCO_EN bit . Even if the TCO_EN bit is 0, the NMI's will still be routed to cause the SMI#. NOTE: This bit can not be written once the TCO_LOCK bit (at offset 08h of TCO I/O Space) is set. This prevents unauthorized software from disabling the generation of TCO-based SMI's. This bit is reset by PLTRST# assertion.
12	0h RO	Reserved.

7.1.17 TCO Base Address (TCOBASE)—Offset 50h

TCO Base Address

Access Method

Type: CFG Register
(Size: 32 bits)

Device: 31
Function: 4

Default: 1h

Bit Range	Default and Access	Field Name (ID): Description
31:16	0h RO	Reserved.
15:5	0h RW/L	**TCO Base Address (TCOBA):** Provides the 32 bytes of I/O space for TCO logic, mappable anywhere in the 64k I/O space on 32-byte boundaries.
4:1	0h RO	Reserved.
0	1h RO	**I/O Space (IOS):** Indicates an I/O Space

- TCO registers are available behind an I/O base address
- TCO_BASE is at offset 0x50 of SMBus PCI register space (device 31 (0x1f) function 4)

34.1.6 TCO1_CNT Register (TCTL1)—Offset 8h

TCO1_CNT Register

Access Method

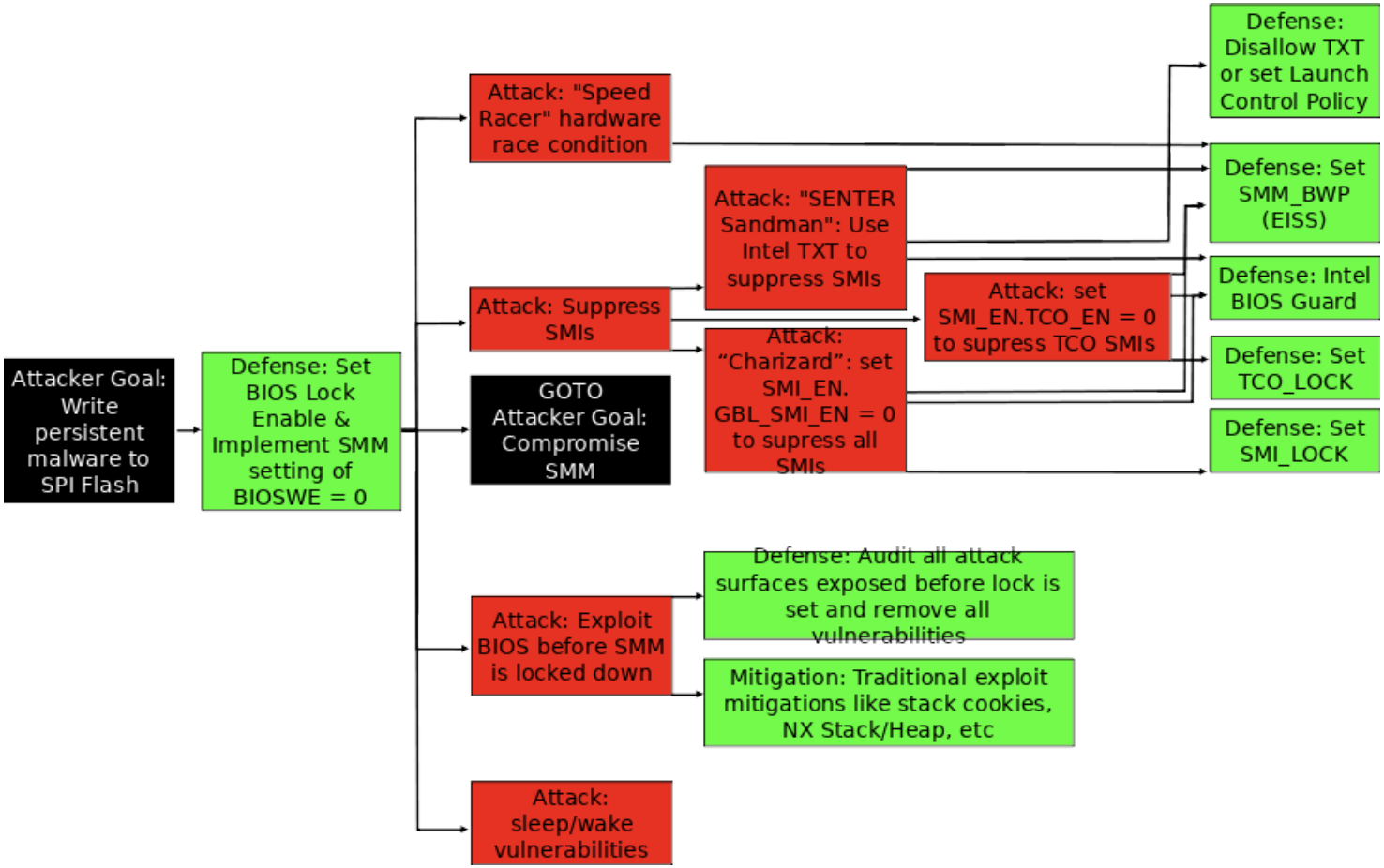
Type: IO Register
(Size: 16 bits)

Device:
Function:

Default: 0h

Bit Range	Default and Access	Field Name (ID): Description
15:13	0h RO	Reserved.
12	0h RW	**TCO_LOCK:** When set to 1, this bit prevents writes from changing the TCO_EN bit (in offset 30h of Power Management I/O space). Once this bit is set to 1, it can not be cleared by software writing a 0 to this bit location. A core-well reset is required to change this bit from 1 to 0. This bit defaults to 0.

- TCO_LOCK is available at TCO_BASE offset 0x8
- TCO_LOCK prevents the TCO_EN bit from being changed in the ACPI SMI_EN register
- It is crucial to set these bits (TCO_EN and TCO_LOCK to 1) in order to make WPD protection effective



- PRR stands for Protected Range Registers
- Chipset registers that can prohibit flash access for specified regions (even in SMM)
- PRR protection, once set and locked, is in place until the system is reset
- PRRs reside behind **SPI BAR**
- **SPI BAR** register is accessed via PCI device 31 (0x1f) function 5 offset 0x10 (standard offset for PCI device BARs)

8.1.5 SPI BAR0 MMIO (BIOS_SPI_BAR0)—Offset 10h

Access Method

Type: CFG Register (Size: 32 bits)

Device: 31
Function: 5

Default: 0h

Bit Range	Default and Access	Field Name (ID): Description
31:12	0h RW	**Memory BAR (MEMBAR):** Software programs this register with the base address of the device's memory region. The Host/BIOS MMIO registers in the flash controller are offset from this BAR.
11:4	0h RO	**Memory Size (MEMSIZE):** Hardwired to 0 to indicate 4KB of memory space
3	0h RO	**Prefetchable (PREFETCH):** A device can mark a range as prefetchable if there are no side effects on reads, the device returns all bytes on reads regardless of the byte enables. Hardwired to 1 to indicate the device's memory space as prefetchable.
2:1	0h RO	**Type (TYP):** Hardwired to 0 to indicate that Base register is 32 bits wide and mapping can be done anywhere in the 32-bit Memory Space.
0	0h RO	**Memory Space Indicator (MEMSPACE):** Hardwired to 0 to identify a Memory BAR.

8.2.29 Flash Protected Range 1 (BIOS_FPR1)—Offset 88h

This register cannot be written when the FLOCKDN bit is set to 1.

Access Method

Type: MEM Register
(Size: 32 bits)

Device:
Function:

Default: 0h

Bit Range	Default and Access	Field Name (ID): Description
31	0h RW/L	**Write Protection Enable (WPE):** When set, this bit indicates that the Base and Limit fields in this register are valid and that writes and erases directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
30:16	0h RW/L	**Protected Range Limit (PRL):** This field corresponds to FLA address bits 26:12 and specifies the upper limit of the protected range. Address bits 11:0 are assumed to be FFFh for the limit comparison. Any address greater than the value programmed in this field is unaffected by this protected range.
15	0h RW/L	**Read Protection Enable (RPE):** When set, this bit indicates that the Base and Limit fields in this register are valid and that reads directed to addresses between them (inclusive) must be blocked by hardware. The base and limit fields are ignored when this bit is cleared.
14:0	0h RW/L	**Protected Range Base (PRB):** This field corresponds to FLA address bits 26:12 and specifies the lower base of the protected range. Address bits 11:0 are assumed to be 000h for the base comparison. Any address less than the value programmed in this field is unaffected by this protected range.

8.2.2 Hardware Sequencing Flash Status and Control (BIOS_HSFSTS_CTL)—Offset 4h

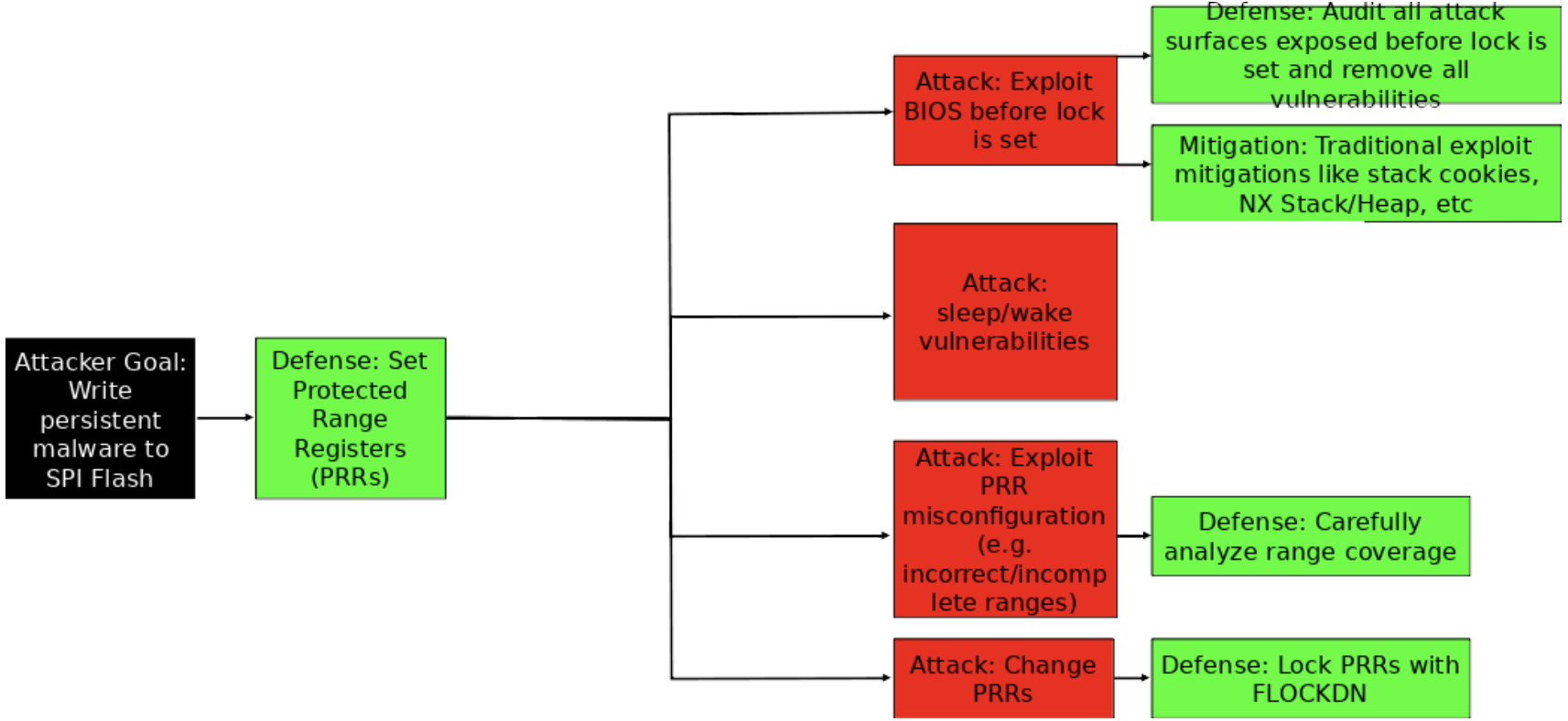
Access Method

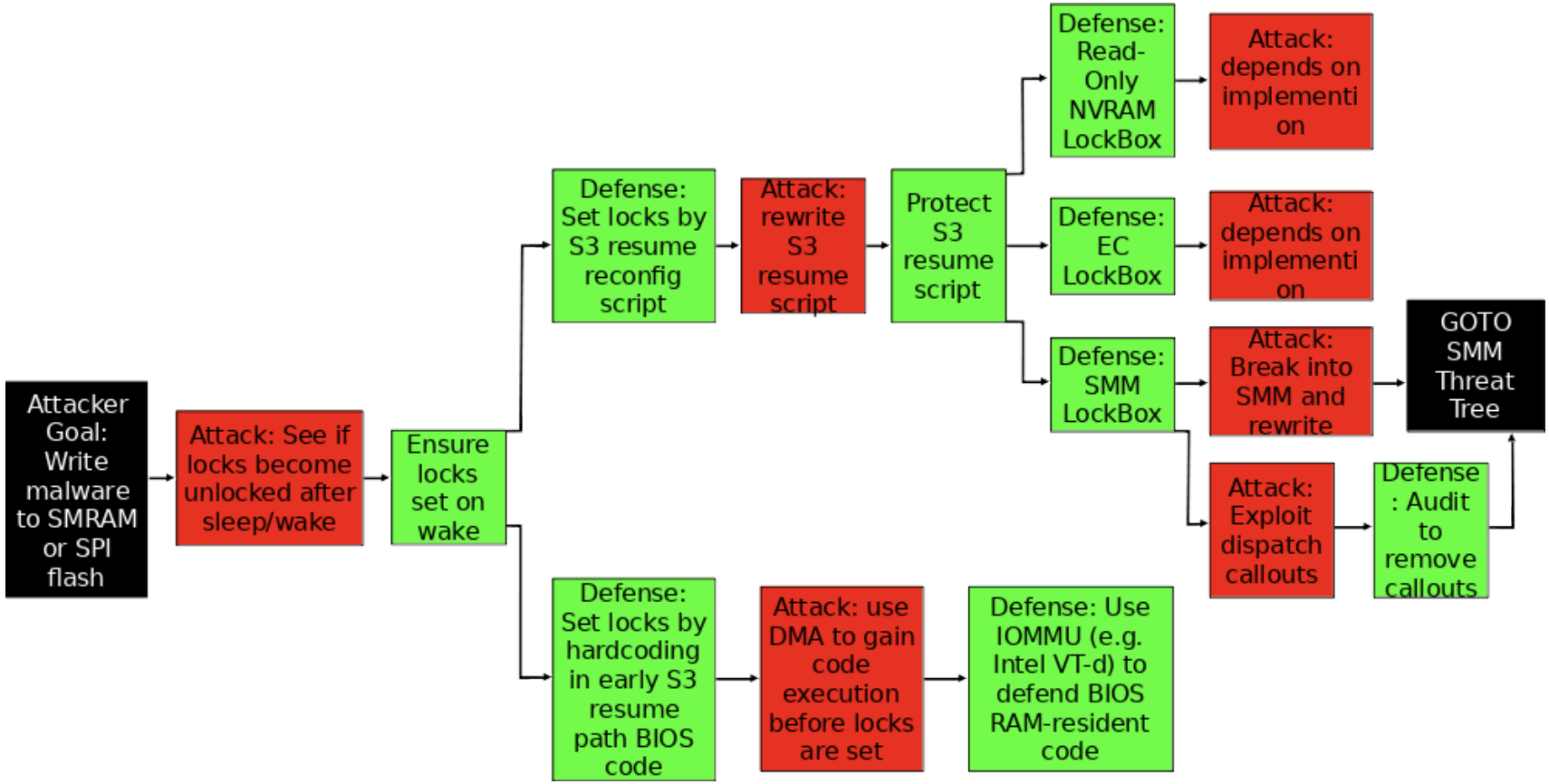
Type: MEM Register
(Size: 32 bits)

Device:
Function:

Default: 2000h

15	0h RW/L	**Flash Configuration Lock-Down (FLOCKDN):** When set to 1, those Flash Program Registers that are locked down by this FLOCKDN bit cannot be written. Once set to 1, this bit can only be cleared by a hardware reset.
14	0h RO/V	**Flash Descriptor Valid (FDV):** This bit is set to a 1 if the Flash Controller read the correct Flash Descriptor Signature. If the Flash Descriptor Valid bit is not 1 , software cannot use the Hardware Sequencing registers, but must use the software sequencing registers. Any attempt to use the Hardware Sequencing registers will result in the FCERR bit being set
13	1h RO/V	**Flash Descriptor Override Pin-Strap Status (FDOPSS):** This register reflects the value the Flash Descriptor Override Pin-Strap. '1': No override '0': The Flash Descriptor Override strap is set





Quiz

Quiz

Where are the flash regions and their access permissions defined?

Quiz

Where are the flash regions and their access permissions defined?

- Intel Flash Descriptor

Quiz

Where are the flash regions and their access permissions defined?

- Intel Flash Descriptor

How can the Flash Descriptor region access permissions be bypassed?

Quiz

Where are the flash regions and their access permissions defined?

- Intel Flash Descriptor

How can the Flash Descriptor region access permissions be bypassed?

- ME HECI HMRFPO_ENABLE command
- Flash Descriptor Override Security Pin Strap
- External programmer used to reprogram Flash Descriptor

Quiz

Where are the flash regions and their access permissions defined?

- Intel Flash Descriptor

How can the Flash Descriptor region access permissions be bypassed?

- ME HECI HMRFPO_ENABLE command
- Flash Descriptor Override Security Pin Strap
- External programmer used to reprogram Flash Descriptor

Can Protected Range Registers (PRRs) be bypassed?

Quiz

Where are the flash regions and their access permissions defined?

- Intel Flash Descriptor

How can the Flash Descriptor region access permissions be bypassed?

- ME HECI HMRFPO_ENABLE command
- Flash Descriptor Override Security Pin Strap
- External programmer used to reprogram Flash Descriptor

Can Protected Range Registers (PRRs) be bypassed?

- Yes, BIOS Guard can bypass PRRs if ME is configured to allow it
- No other methods are known except faulty BIOS implementations not setting them back during S3 resume)

Conclusion

- We know why adversaries like to target BIOS flash
- We have learned about BIOS flash protection mechanisms offered by Intel chipsets/PCH
- We have learned about SPI flash vendor security features on the example of Winbond W25Q128FV

Q&A

Slim Bootloader Measured Boot

Architecture 5141

Goals of the Presentation

In this presentation, we aim to:

- Discuss requirements for Measured Boot in context of Slim Bootloader and ODROID-H4.
- Explain PCRs and TCG Event Log and how those could be used during attestation.
- Look at implementation of measured boot on UEFI BIOS and Slim Bootlaoder
- Deep dive into code and bootlog of Dasharo (Slim Bootloader + UEFI) to understand how measured boot works.
- Analyzes Event Log for AMI BIOS and Dasharo (Slim Bootlaoder + UEFI) in context of boot states.
- Discuss production approach.

Recommended Materials

- Slim Bootloader documentation
- TC1101: Introductory Trusted Platform Module (TPM) usage
- TC1102: Intermediate Trusted Platform Module (TPM) usage
- A Tour Beyond BIOS: Implementing TPM2 Support in EDKII

Measured boot requirements

Measured boot requirements

- Root of Trust for Measurement

Measured boot requirements

- Root of Trust for Measurement
- Root of Trust for Reporting

Measured boot requirements

- Root of Trust for Measurement
- Root of Trust for Reporting
- Root of Trust for Storage (including confidentiality and integrity)

Measured boot requirements

- Root of Trust for Measurement
- Root of Trust for Reporting
- Root of Trust for Storage (including confidentiality and integrity)
- All those requirements can e filled by TPM.

dTPM vs fTPM

- Discrete TPM, or dTPM, is a separate component that is physically connected onto the motherboard to provide hardware-based encryption.
- Integrated TPM (iTPM) solution, using dedicated hardware integrated into one or more semiconductor packages alongside, but logically separate from, other components.
- Firmware TPM solution, running the TPM in firmware in a Trusted Execution mode of a general purpose computation unit.
- Leading semiconductor manufacturers offer fTPM within their later generation chipsets (Intel) or CPUs/SoCs (AMD) to allow for additional protection and convenience without the need for a separate physical module.
- Typically UEFI BIOS contain option in setup for enabling/disabling dTPM/fTPM
- 3mdeb blog post

- dTPMs
  - typically FIPS certified
  - better suited to large scale organization commision/decomission processes
  - compliant to TCG profiles
- fTPMs
  - low value, every day use cases
  - often not compliant to TCG profiles, often just minimum to make Windows happy
  - Intel PTT (Platform Trust Technology) and AMD fTPM are just software in crypto-cooprocessor (CSME, PSP/ASP)
  - resistant to TPM Genie MiTM attack's

TPM1.2 vs TPM2.0 vs TCM

- Because of Microsoft mandate TPM1.2 are disappearing from the market.
- Despite that TPM1.2 still can be used as valid root of trust for some purposes when you are aware of limitations.
- Main problem is use of SHA1 which is these days considered weak hash.
- TPM1.2 can be leveraged for Measured Boot
- Chinese government defines Trusted Cryptography Module (TCM)
  - TCM provides the root-of-trust for reporting (RTR) and the root-of-trust for storage (RTS) and supports the trusted boot. It is similar to TPM from the functionality perspective.
  - For example, TCM defines the Platform Configuration Register (PCR), non-volatile memory, endorsement key, identity key, and so on. TCM only supports Chinese cryptography – SM2, SM3, and SMS4.

TPM PCRs

Platform Configuration Registers

Formula for populating PCR with new measurement is as follows:

$$PCR_{(new)} = HASH\ (PCR_{(old)}\ ||\ HASH(Data))$$

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

PCR Index	PCR Usage
16	Debug
23	Application Support

TCG PC Client Platform Firmware Profile Specification Version 1.06 Revision 52

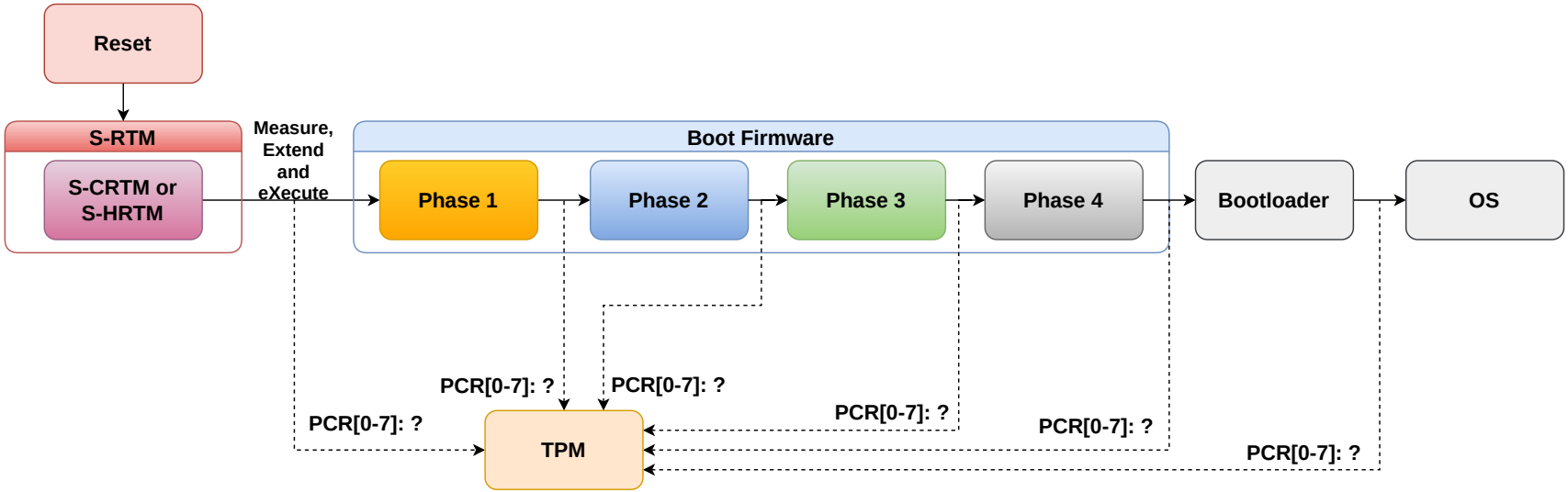
Measured boot photo *Garaolaza, CC BY-SA 4.0, via Wikimedia Commons*

- Conceptually different than verified boot
- Means calculating hashes of the loaded code and data bits and storing them in a secure place

Measured Boot

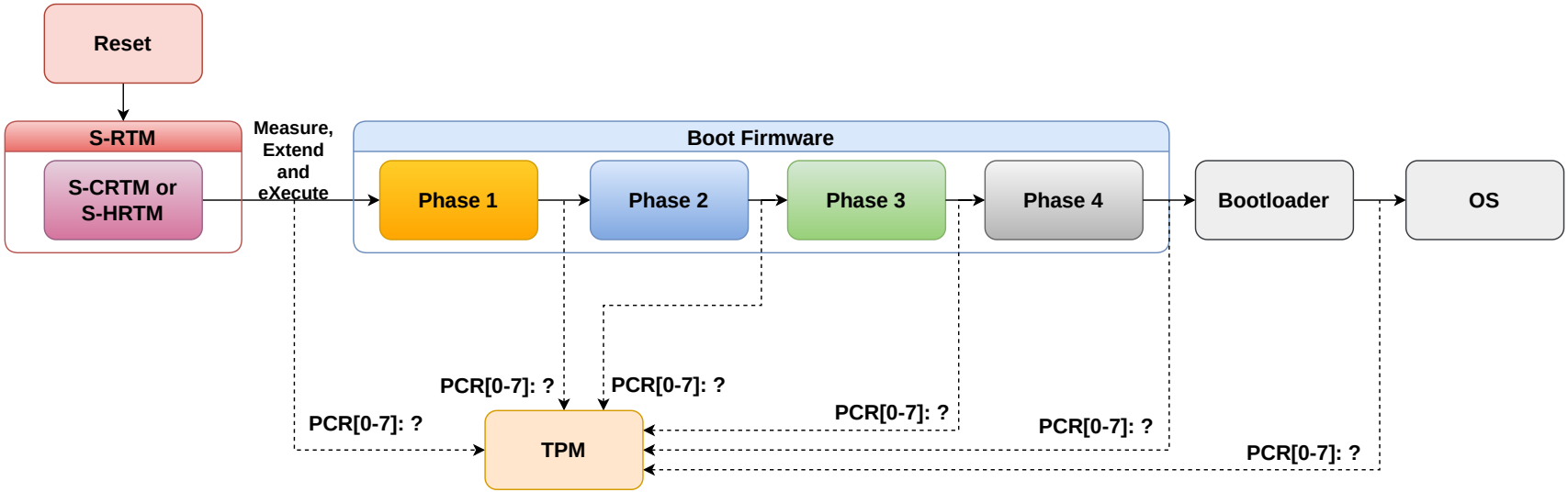
- Also called Trusted Boot.
- It is a Chain of Trust method.
- Measured Boot does not fail because there is no verification during boot process.
- After finishing boot process other software makes a security decision.
- That decision is called attestation.
- There are two types of attestation: local and remote.

How Measured Boot works?



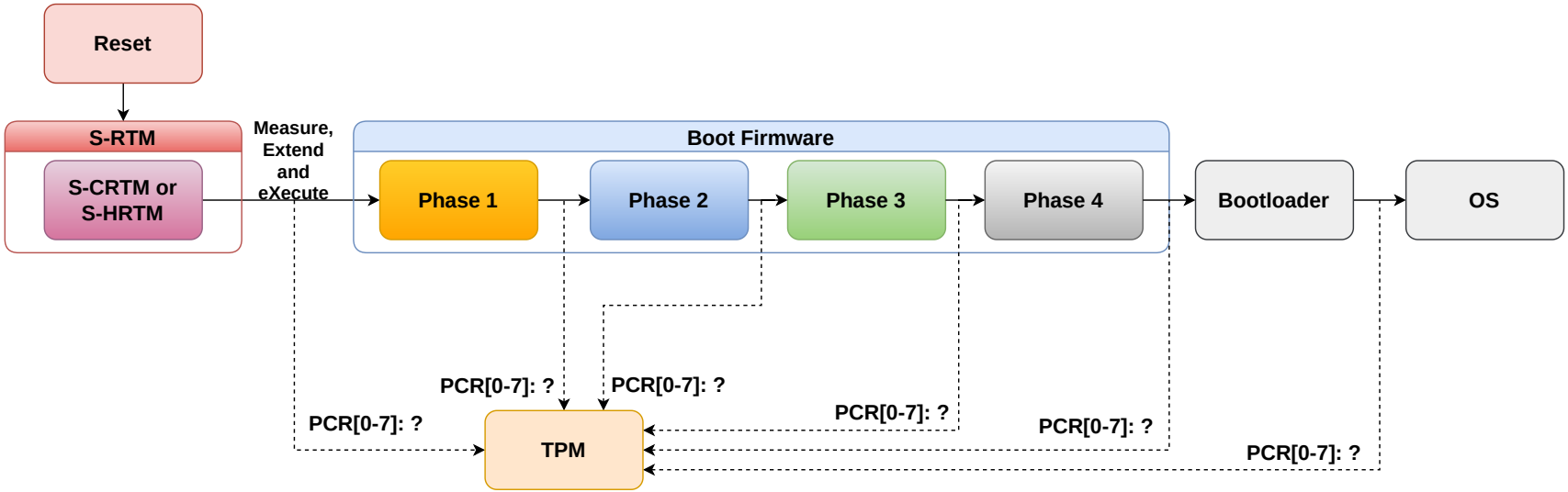
- **Measured Boot** - is boot process during which transitive trust was performed and chain of trust (chain of integrity measurements) was established.

How Measured Boot works?



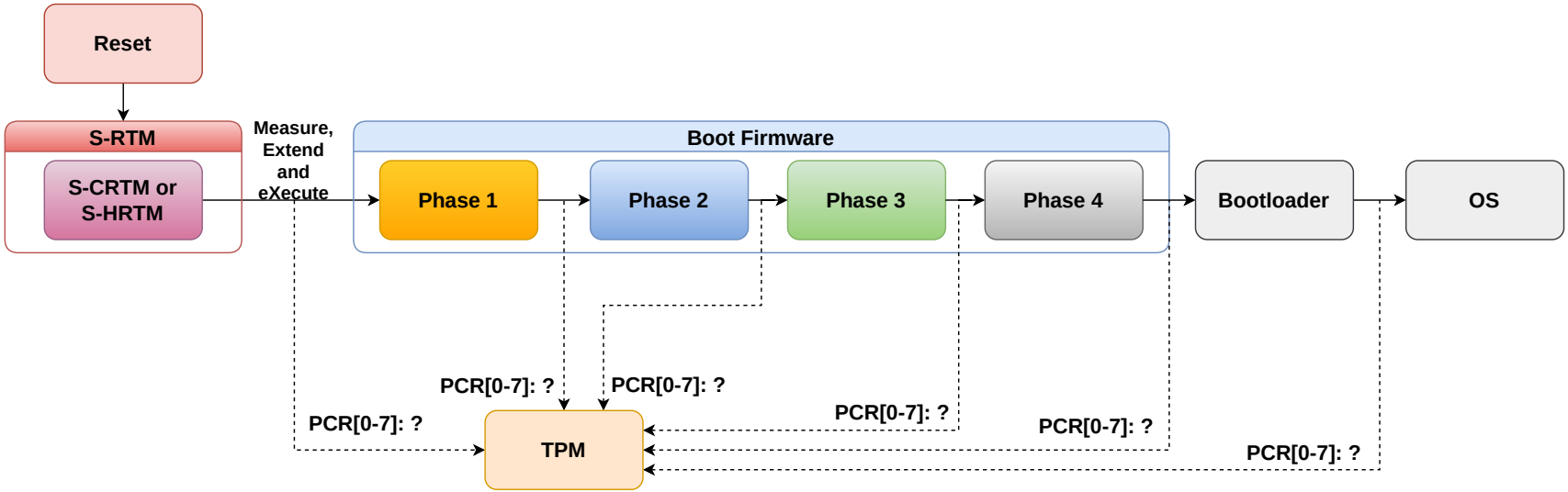
- **Measured Boot** - is boot process during which transitive trust was performed and chain of trust (chain of integrity measurements) was established.

How Measured Boot works?



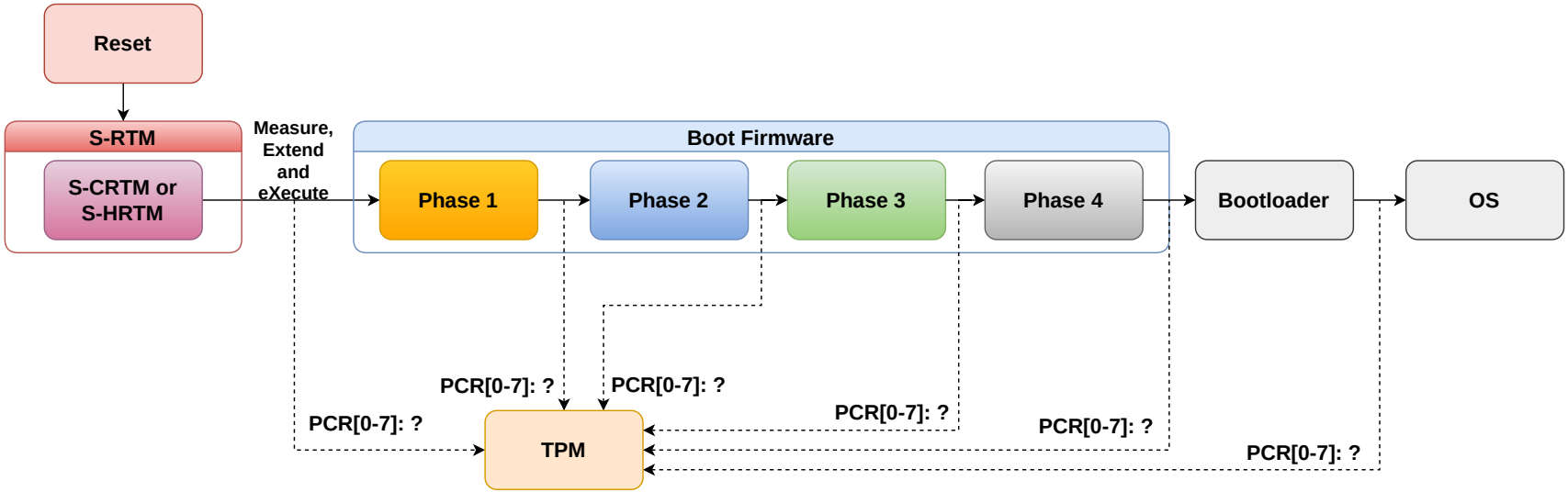
- **Measured Boot** - is boot process during which transitive trust was performed and chain of trust (chain of integrity measurements) was established.

How Measured Boot works?



- **Measured Boot** - is boot process during which transitive trust was performed and chain of trust (chain of integrity measurements) was established.

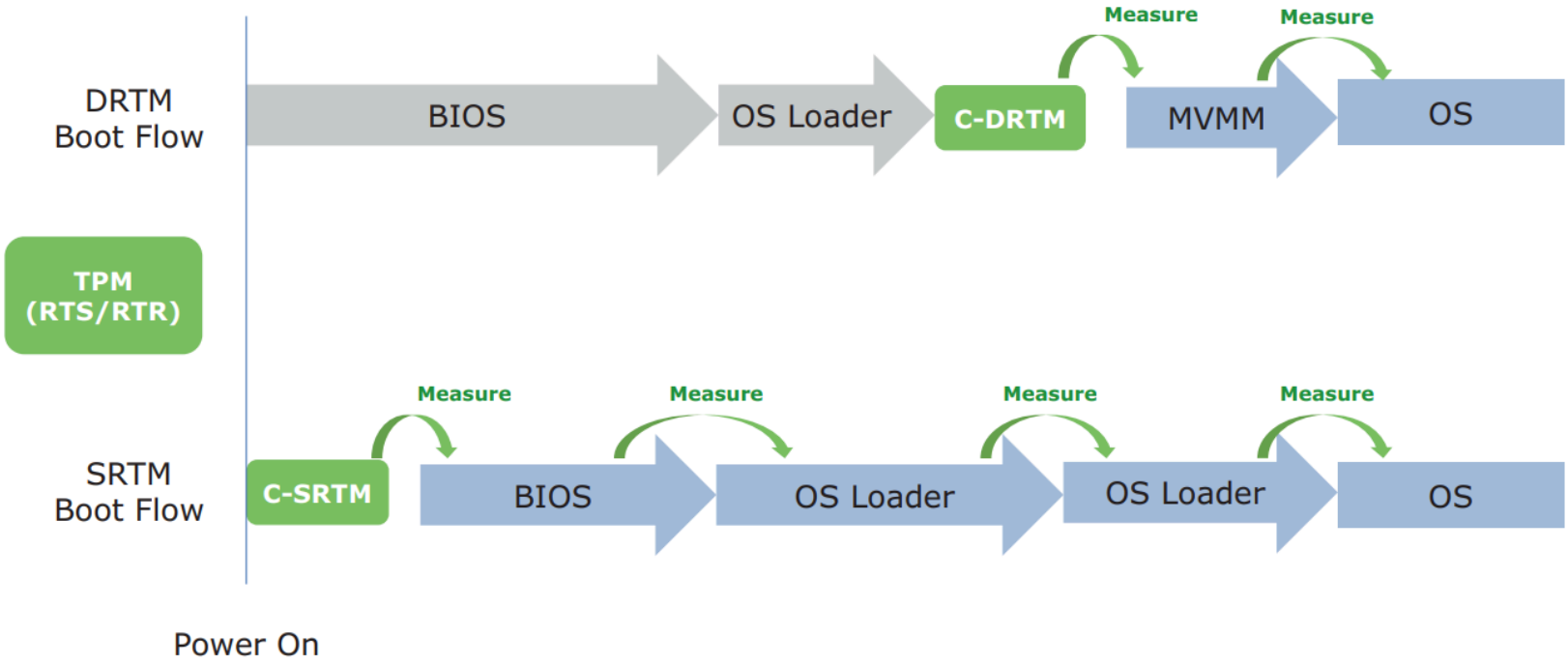
How Measured Boot works?



- **Measured Boot** - is boot process during which transitive trust was performed and chain of trust (chain of integrity measurements) was established.

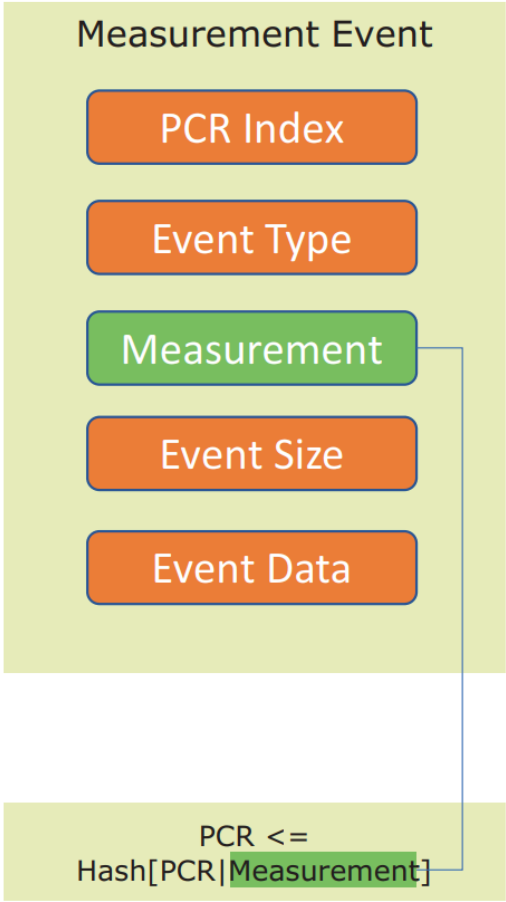
How to protect S-RTM?

- TL;DR It depends on the mechanism delivered by the platform and/or silicon vendor.
- Verified Boot-like technologies with strong RTV (Intel Boot Guard, AMD Hardware Validated Boot, NXP High Assurance Boot, etc.)
- How hard is it to sign firmware and fuse/provision platform? It depends.
- What we can do with all those measurements in TPM?
  - local attestation,
  - remote attestation,
- Maybe Measured Boot and S-RTM don’t make sense, and Verified Boot is the one to rely on?
  - ownership transfer,
  - vendor lock-in,
  - closed, centralized authority vs open, decentralized community,
- As always combining SV RTV+S-CRTM with SV CoT, firmware framework CoT and industry standard OS interface gives best security.

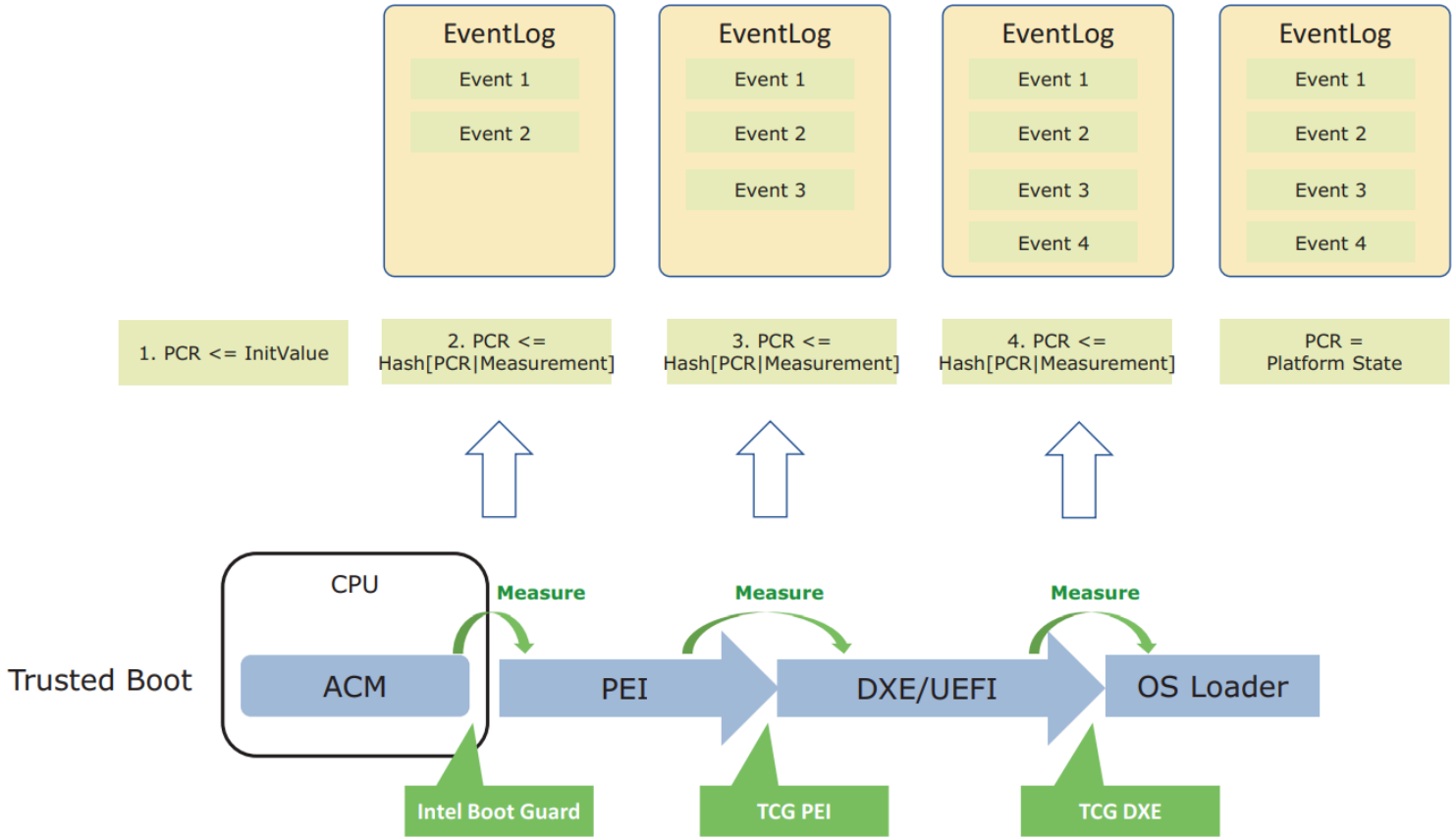


"Building Secure Firmware", Jiewen Yao, Vincent Zimmer, 2020

- The PCRs hold the final hash of the measured components (code and/or data).
- It is unfeasible to extract individual measurements from the final PCR value.
- TCG defines the integrity measurement **event log** to record the individual measurement hashes and the description for the measurement.
- `PCR Index` - PCR number to which `Measurement` was extended,
- `Event Type` - what was measured, there are over 30 types described in TCG spec, but in essence it identifies what was measured UEFI variable, PE/COFF file, microcode etc.
- `Measurement` - digest of component,
- `Event Size` - length in bytes `Event Data`
- `Event Data` - structure defined by `Event Type`



"Building Secure Firmware", Jiewen Yao, Vincent Zimmer, 2020



"Building Secure Firmware", Jiewen Yao, Vincent Zimmer, 2020

Attestation

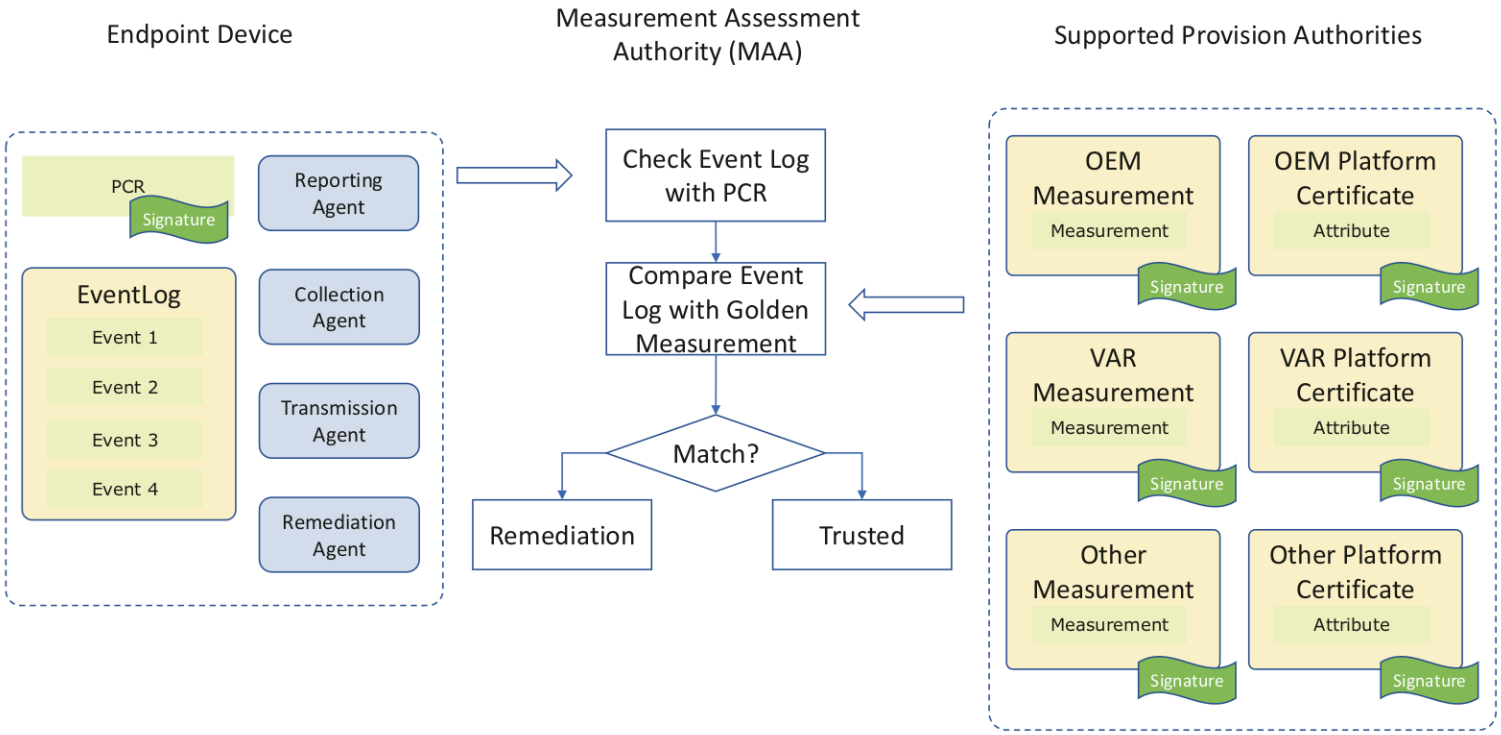
- Process of proving TPM PCR values can be internal to TPM or external.
- Internal attestation, also called local attestation, is essentially execution of authorization based on configured policy which decides if Sealed Data Object would be provided by TPM.
- TPM sealed data is small: 128 bytes.
- External attestation, also called remote attestation, is more complex and requires creation of key hierarchy, which will sign measurements from TPM, as well as attestation server which can make decisions based on measurements.

Unsealing

- "Unsealing" is the term for TPM returning bytes that were previously stored inside a TPM-created sealed object, it is requested using `TPM2_Unseal` command.
- The only way the TPM will return those bytes is if the authorization policy embedded in that object is satisfied.
- A very common policy is "PolicyPCR with these PCRs," so the gate becomes "current measured boot state must match what the object was created for."
- The check and the decision happen inside the TPM.
- If the policy is satisfied, the TPM decrypts and integrity-checks the object's private blob using its storage key hierarchy and emits the plaintext secret on the TPM command response; if not, it returns an error and no secret leaves the chip.
- The returned value may be encrypted using authorization session encryption.

Remote attestation

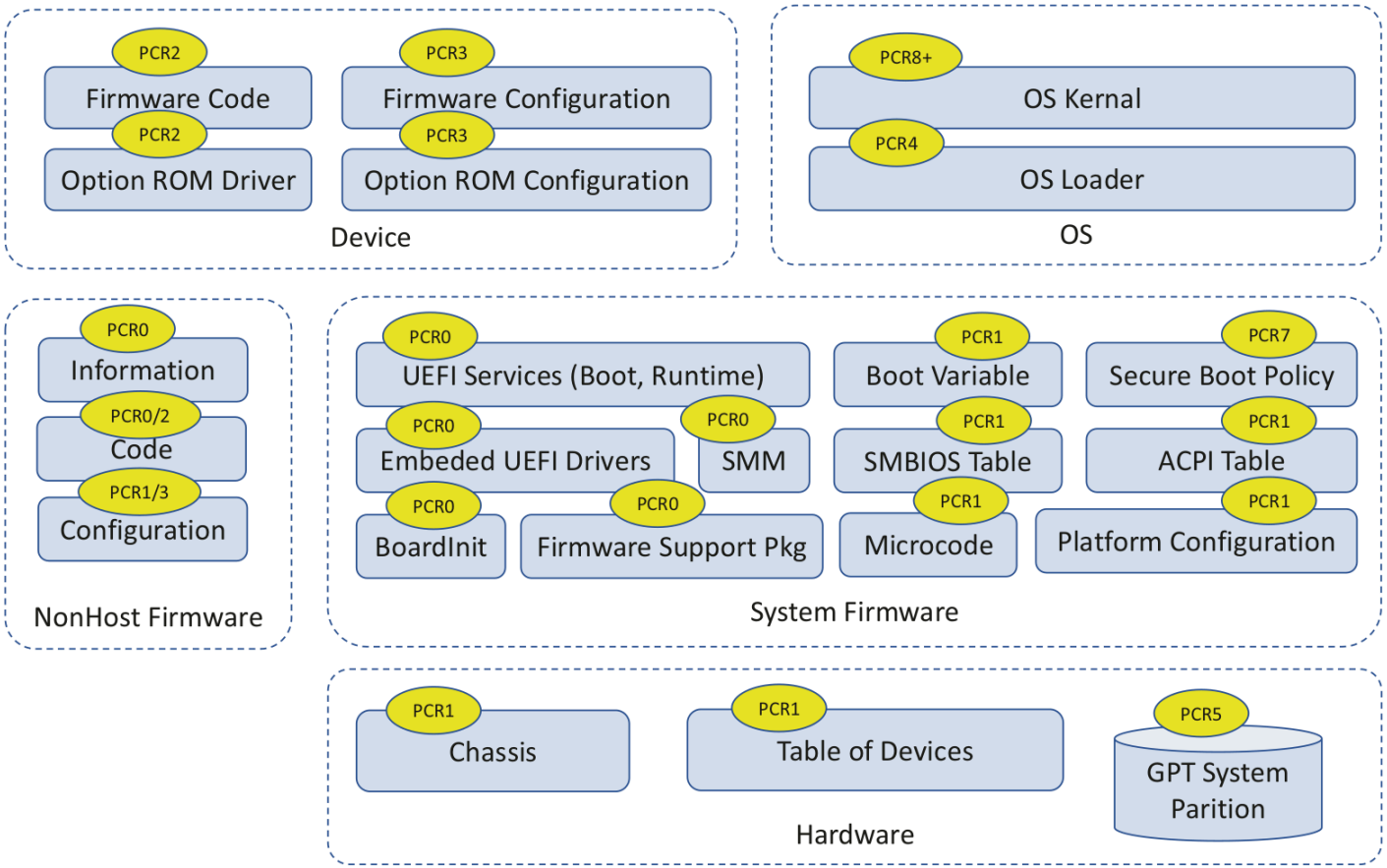
- We recognize two parties: attester and verifier.
- The `TPM_Quote` is a command to sign a set of PCR values with the TPM private key and provide the digital signature.
- It is leveraged when PCR values have to be transferred through untrusted environment (network, software stack).
- Verifier performs the verification of the digital signature and the collected PCR values, by comparing with known good one.
- For detailed attestation event log is used. In that case verifier reconstructs PCR values first, based on provided event log.
- Verifier may have multiple policies allowing or disallowing certain actions depending on the attestation result.
- The complete remote attestation also includes a process to verify the TPM by using the TPM endorsement key (EK).



"Building Secure Firmware", Jiewen Yao, Vincent Zimmer, 2020

Implementation leveraging Measured Boot

- TrustedGRUB2
- LUKS2
- UEFI BIOS
- coreboot
- Slim Bootloader
- Windows BitLocker
- GRUB2
- Linux IMA
- OpenPOWER Trusted Boot
- Project Cerberus
- Microsoft Azure
- Cisco Trust Anchor
- Intel Boot Guard



"Building Secure Firmware", Jiewen Yao, Vincent Zimmer, 2020

Slim Bootloader

- Slim Bootloader implements measured boot supporting both dTPM and fTPM
- It is recommended to enable measured boot in conjunction with verified boot for best firmware security.

```
user@OST2-VM:~$ grep MEASURED_BOOT Platform/AlderlakeBoardPkg/BoardConfigOdroidH4.py -B6
        self.FLASH_BASE_SIZE      = 0x01000000  # 16MB
        self.FLASH_BASE_ADDRESS   = (self.FLASH_LAYOUT_START - self.FLASH_BASE_SIZE)
        self.LOADER_ACPI_RECLAIM_MEM_SIZE = 0x000090000
        self.HAVE_FIT_TABLE       = 1
        self.HAVE_VBT_BIN         = 1
        self.HAVE_VERIFIED_BOOT   = 1
        self.HAVE_MEASURED_BOOT   = 1
--
        self.ENABLE_FAST_BOOT = 0
        if self.ENABLE_FAST_BOOT:
            self.ENABLE_SPLASH              = 0
            self.ENABLE_FRAMEBUFFER_INIT    = 0
            self.RELEASE_MODE               = 1
            self.HAVE_VERIFIED_BOOT         = 0
            self.HAVE_MEASURED_BOOT         = 0
```

```
user@OST2-VM:~$ grep PcdMeasuredBoot **/*.dsc -r -A1 -B2
BootloaderCorePkg/BootloaderCorePkg.dsc-[PcdsFeatureFlag]
BootloaderCorePkg/BootloaderCorePkg.dsc-  gPlatformCommonLibTokenSpaceGuid.PcdMinDecompression    | FALSE
BootloaderCorePkg/BootloaderCorePkg.dsc:  gPlatformCommonLibTokenSpaceGuid.PcdMeasuredBootEnabled | $(HAVE_MEASURED_BOOT)
BootloaderCorePkg/BootloaderCorePkg.dsc-  gPlatformCommonLibTokenSpaceGuid.PcdVerifiedBootEnabled | $(HAVE_VERIFIED_BOOT)
--
BootloaderCorePkg/BootloaderCorePkg.dsc-[PcdsDynamicDefault]
BootloaderCorePkg/BootloaderCorePkg.dsc-  gEfiMdePkgTokenSpaceGuid.PcdPlatformBootTimeOut         | 2
BootloaderCorePkg/BootloaderCorePkg.dsc:  gPlatformCommonLibTokenSpaceGuid.PcdMeasuredBootHashMask | 0x00000002
```

```
#define MEASURED_BOOT_ENABLED()  (FeaturePcdGet (PcdMeasuredBootEnabled)
                                  && (GetFeatureCfg() & FEATURE_MEASURED_BOOT)
                                  && ACPI_FEATURE_ENABLED())
```

- `MEASURED_BOOT_ENABLED()` is the macro that combines multiple conditions to determine if measured boot should be active:
  - The PCD flag must be enabled - our `PcdMeasuredBootEnabled`
  - The FEATURE_MEASURED_BOOT bit must be set in the feature configuration.
  - ACPI must be enabled (as a dependency)
- First place where measurement is needed is Stage1B, because measurement of Stage1A and Stage1B should be made by Intel Boot Guard.

```
/* BootloaderCommonPkg/Include/Library/TpmLib.h */
RETURN_STATUS
EFIAPI
TpmHashAndExtendPcrEventLog (
IN         TPMI_DH_PCR               PcrHandle,
IN         UINT8                     *Data,
IN         UINT32                    Length,
IN         TCG_EVENTTYPE             EventType,
IN         UINT32                    EventSize,
IN  CONST  UINT8                     *Event
);
/* ( ... ) */
RETURN_STATUS
EFIAPI
TpmExtendPcrAndLogEvent (
  IN         TPMI_DH_PCR               PcrHandle,
  IN         TPMI_ALG_HASH             HashAlg,
  IN  CONST  UINT8                     *Hash,
  IN         TCG_EVENTTYPE             EventType,
  IN         UINT32                    EventSize,
  IN  CONST  UINT8                     *Event
  );
```

Practice #310 Exercise #1

- Let's build `DEBUG` version of Dasharo (Slim Bootloader + UEFI) with measured boot enabled (this is default).
- Run `minicom` as usual with logging so we gather output for further analysis.
- Stage1A does not contain any Slim Bootloader Measured Boot related logs, because it does not contain correct infrastructure for the operation (runs from cache). It is also measured and verified by Intel Boot Guard.

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

```
( ... ) | SecStartup2→ContinueFunc→BoardInit(PostTempRamExit)→TmpInitialize→TpmInit
TPM2Startup: TPM_RC_SUCCESS                                  ├→ Tpm2Startup
Supported PCRs - Count = 00000004                            |
GetSupportedAndActivePcrs - Count = 00000001                 ├→ Tpm2GetCapabilitySupportedAndActivePcrs
TpmHashAlgorithmBitmap 0×00000017 ActivePcrBanks 0×00000002  |
TPM Lib Private Data not found                               ├→ TpmLibGetPrivateData
Bootloader requested PCR Bank is enabled.                    ├→ TpmPcrBankCheck
TCG Event Log created at 0×457F2000                          ├→ TmpTcgLogInit
TPM initialization completed succesfully.                    ├→ exit from TpmInit to TmpInitialize
PCR (0) extended successfully with (8) event type.           ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2041                |
HASH Extended B0DEF27F57C94CB9                               ├→ AddEventTCGLog
( ... )                                                      | exit TmpInitialize- and BoardInit
ContinueFunc :cfg data : 0×FFDFB000 length : 0×0             ├→ ContinueFunc
PCR (1) extended successfully with (10) event type.          ├→ TpmExtendPcrAndLogEvent
Adding event in TCG event log at : 0×457F2094                |
HASH Extended B513FAB9B0719A75                               ├→ AddEventTCGLog
ContinueFunc :Hash : 0×FFDFF380 length : 0×0                 |
PCR (0) extended successfully with (2147483656) event type.  |
Adding event in TCG event log at : 0×457F20D6                |
HASH Extended 4BC83838745DDD25                               ├→ CotinueFunc extend KeyHash
( ... )
```

Practice #310 Exercise #2

Let's boot to DTS and copy TPM2 EventLog:

```
user@OST2-HW:~# cp /sys/kernel/security/tpm0/binary_bios_measurements eventlog
```

Then parse it using on of the program included in `tpm2-tools`:

```
user@OST2-HW:~# tpm2_eventlog eventlog | less
```

```
version: 1
events:
- EventNum: 0
  PCRIndex: 0
  EventType: EV_NO_ACTION
  Digest: "0000000000000000000000000000000000000000"
  EventSize: 33
  SpecID:
  - Signature: Spec ID Event03
    platformClass: 0
    specVersionMinor: 0
    specVersionMajor: 2
    specErrata: 0
    uintnSize: 1
    numberOfAlgorithms: 1
    Algorithms:
    - Algorithm[0]:
      algorithmId: sha256
      digestSize: 32
    vendorInfoSize: 0
```

```
- EventNum: 1
  PCRIndex: 0
  EventType: EV_S_CRTM_VERSION
  DigestCount: 1
  Digests:
  - AlgorithmId: sha256
    Digest: "b94cc9577ff2deb0de5fd53255ac421d61f0c3cfa69bd38b47346065f859155f"
  EventSize: 33
  Event: "2453424821000100534 25f41444c4e20000000090000010120000000000000000"
```

```
- EventNum: 2
  PCRIndex: 1
  EventType: EV_PLATFORM_CONFIG_FLAGS
  DigestCount: 1
  Digests:
  - AlgorithmId: sha256
    Digest: "759a71b0b9fa13b5f68cfe046cafa890f8492c99bad4a964a0f91495e56662e1"
  EventSize: 16
  Event: "00b0dfff000000004c08000000000000"
```

```
- EventNum: 3
  PCRIndex: 0
  EventType: EV_EFI_PLATFORM_FIRMWARE_BLOB
  DigestCount: 1
  Digests:
  - AlgorithmId: sha256
      Digest: "25dd5d743838c84b750eed582d6e63e5d8fa0150f3093bca59f392f223896e9d"
  EventSize: 16
  Event:
    BlobBase: 0×ffdff380
    BlobLength: 0×108
```

```
- EventNum: 4
  PCRIndex: 0
  EventType: EV_EFI_PLATFORM_FIRMWARE_BLOB
  DigestCount: 1
  Digests:
  - AlgorithmId: sha256
    Digest: "d80155cd4ee1736cea35015aa521b093c3c030763f94d46e53a5cffc3bc98ce8"
  EventSize: 16
  Event:
    BlobBase: 0×ffd19000
    BlobLength: 0×7fe6c
```

```
- EventNum: 5
  PCRIndex: 0
  EventType: EV_EFI_PLATFORM_FIRMWARE_BLOB
  DigestCount: 1
  Digests:
  - AlgorithmId: sha256
    Digest: "c472b363d965a57199b65129af43e8c64360007d514ac738a07e72359f9b186f"
  EventSize: 16
  Event:
    BlobBase: 0×ff6ba370
    BlobLength: 0×1771c8
pcrs:
  sha256:
    0  : 0×7df967f4a83a62bc76cd9fde2c0d4d8d5a217c7e17aabf58a105756e63d719a9
    1  : 0×96f1be53c82a36a16e46a8588e934293c62dfc843ff22987fc1506ca83455b06
```

- AMI BIOS event log contains 35 events in 9 PCRs (PCR0-7 and PCR9)
- Not all can be clearly mapped.
- Different convention used:
  - initial event contain revision as UEFI spec errata
  - microcode is measured
  - Secure Boot, KEK and dbx are measured
  - vendor-specific variables are measured

```
pcrs:
  sha256:
    0  : 0×60813c8d3e14ba60838c5d19331878165267f6fc900dc30acfc92180c20258ed
    1  : 0×5c8824c7a0240b8cf73009666ec76170fa33173c318c45e9c7cf12463a1c8621
    2  : 0×6fd2ee8c9f5eb6d6ecaf179121acc25610b78c00495ba6f484b063a3db7dcc3e
    3  : 0×3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    4  : 0×0b3b1d720377fde087297f2be45bf82a8cc766e8e737380f507fab19fd70e941
    5  : 0×e3be491dfd2671e74996a101705e3d7f8bbdab744a310b546e496a60f09e1472
    6  : 0×3d458cfe55cc03ea1f443f1562beec8df51c75e14a9fcf9a7234a13f198e7969
    7  : 0×7d6a93b8faca035859ac72881cfad841a9935c40a45f942c67927ed3279a9d68
    9  : 0×dcb3244aa4f6485384864fc7f627553ff6a91ce417d273ab2512d462edf6f952
```

- Let's confirm we have the same set of measurements for AMD and Dasharo (Slim Bootloader + UEFI) every time on various boot paths:
  - soft reset (`reboot` command)
  - hard reset (reset button)
  - cold boot (soft off: `poweroff` aka ACPI S5)
  - S3 (only for AMI)

```
user@OST2-VM:~# tpm2_pcrread
  sha1:
  sha256:
    0 : 0x60813C8D3E14BA60838C5D19331878165267F6FC900DC30ACFC92180C20258ED
    1 : 0x59C4A67EB2FB00362EA2DBF84AF684513730F6B28B33C73DD86D15C15324A4EF
    2 : 0x6FD2EE8C9F5EB6D6ECAF179121ACC25610B78C00495BA6F484B063A3DB7DCC3E
    3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    4 : 0x0B3B1D720377FDE087297F2BE45BF82A8CC766E8E737380F507FAB19FD70E941
    5 : 0xE3BE491DFD2671E74996A101705E3D7F8BBDAB744A310B546E496A60F09E1472
    6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    7 : 0x7D6A93B8FACA035859AC72881CFAD841A9935C40A45F942C67927ED3279A9D68
    8 : 0x0000000000000000000000000000000000000000000000000000000000000000
    9 : 0xDCB3244AA4F6485384864FC7F627553FF6A91CE417D273AB2512D462EDF6F952
    10: 0x0000000000000000000000000000000000000000000000000000000000000000
    11: 0x0000000000000000000000000000000000000000000000000000000000000000
    12: 0x0000000000000000000000000000000000000000000000000000000000000000
    13: 0x0000000000000000000000000000000000000000000000000000000000000000
    14: 0x0000000000000000000000000000000000000000000000000000000000000000
    15: 0x0000000000000000000000000000000000000000000000000000000000000000
    16: 0x0000000000000000000000000000000000000000000000000000000000000000
    17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    23: 0x0000000000000000000000000000000000000000000000000000000000000000
```

```
user@OST2-VM:~# tpm2_pcrread
  sha1:
  sha256:
    0 : 0x60813C8D3E14BA60838C5D19331878165267F6FC900DC30ACFC92180C20258ED
    1 : 0xAFFAF8271112CCCEBEDB9C50CD8AEC4BDD4F8809416C9388F00C074DD2C88DAD
    2 : 0x6FD2EE8C9F5EB6D6ECAF179121ACC25610B78C00495BA6F484B063A3DB7DCC3E
    3 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    4 : 0x0B3B1D720377FDE087297F2BE45BF82A8CC766E8E737380F507FAB19FD70E941
    5 : 0xE3BE491DFD2671E74996A101705E3D7F8BBDAB744A310B546E496A60F09E1472
    6 : 0x3D458CFE55CC03EA1F443F1562BEEC8DF51C75E14A9FCF9A7234A13F198E7969
    7 : 0x7D6A93B8FACA035859AC72881CFAD841A9935C40A45F942C67927ED3279A9D68
    8 : 0x0000000000000000000000000000000000000000000000000000000000000000
    9 : 0xDCB3244AA4F6485384864FC7F627553FF6A91CE417D273AB2512D462EDF6F952
    10: 0x0000000000000000000000000000000000000000000000000000000000000000
    11: 0x0000000000000000000000000000000000000000000000000000000000000000
    12: 0x0000000000000000000000000000000000000000000000000000000000000000
    13: 0x0000000000000000000000000000000000000000000000000000000000000000
    14: 0x0000000000000000000000000000000000000000000000000000000000000000
    15: 0x0000000000000000000000000000000000000000000000000000000000000000
    16: 0x0000000000000000000000000000000000000000000000000000000000000000
    17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
    23: 0x0000000000000000000000000000000000000000000000000000000000000000
```

S3 testing

```
user@OST2-HW:~# echo deep | tee /sys/power/mem_sleep >/dev/null
user@OST2-HW:~# rtcwake -m mem -s 60
```

- Measurements are the same as on from the state we trigger S3.

```
sha1:
sha256:
  0 : 0x7DF967F4A83A62BC76CD9FDE2C0D4D8D5A217C7E17AABF58A105756E63D719A9
  1 : 0x96F1BE53C82A36A16E46A8588E934293C62DFC843FF22987FC1506CA83455B06
  2 : 0x0000000000000000000000000000000000000000000000000000000000000000
  3 : 0x0000000000000000000000000000000000000000000000000000000000000000
  4 : 0x0000000000000000000000000000000000000000000000000000000000000000
  5 : 0x0000000000000000000000000000000000000000000000000000000000000000
  6 : 0x0000000000000000000000000000000000000000000000000000000000000000
  7 : 0x0000000000000000000000000000000000000000000000000000000000000000
  8 : 0x0000000000000000000000000000000000000000000000000000000000000000
  9 : 0x0000000000000000000000000000000000000000000000000000000000000000
  10: 0x0000000000000000000000000000000000000000000000000000000000000000
  11: 0x0000000000000000000000000000000000000000000000000000000000000000
  12: 0x0000000000000000000000000000000000000000000000000000000000000000
  13: 0x0000000000000000000000000000000000000000000000000000000000000000
  14: 0x0000000000000000000000000000000000000000000000000000000000000000
  15: 0x0000000000000000000000000000000000000000000000000000000000000000
  16: 0x0000000000000000000000000000000000000000000000000000000000000000
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000000000000000000000000000
sha384:
sm3_256:
```

```
sha1:
sha256:
  0 : 0x7DF967F4A83A62BC76CD9FDE2C0D4D8D5A217C7E17AABF58A105756E63D719A9
  1 : 0x96F1BE53C82A36A16E46A8588E934293C62DFC843FF22987FC1506CA83455B06
  2 : 0x0000000000000000000000000000000000000000000000000000000000000000
  3 : 0x0000000000000000000000000000000000000000000000000000000000000000
  4 : 0x0000000000000000000000000000000000000000000000000000000000000000
  5 : 0x0000000000000000000000000000000000000000000000000000000000000000
  6 : 0x0000000000000000000000000000000000000000000000000000000000000000
  7 : 0x0000000000000000000000000000000000000000000000000000000000000000
  8 : 0x0000000000000000000000000000000000000000000000000000000000000000
  9 : 0x0000000000000000000000000000000000000000000000000000000000000000
  10: 0x0000000000000000000000000000000000000000000000000000000000000000
  11: 0x0000000000000000000000000000000000000000000000000000000000000000
  12: 0x0000000000000000000000000000000000000000000000000000000000000000
  13: 0x0000000000000000000000000000000000000000000000000000000000000000
  14: 0x0000000000000000000000000000000000000000000000000000000000000000
  15: 0x0000000000000000000000000000000000000000000000000000000000000000
  16: 0x0000000000000000000000000000000000000000000000000000000000000000
  17: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  18: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  19: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  20: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  21: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  22: 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
  23: 0x0000000000000000000000000000000000000000000000000000000000000000
sha384:
sm3_256:
```

Homework

- Do slight modification in BoardConfig and prove measurements changed.
- Perform full reconstruction of PCRs.
- Show how PCRs differ when IBG is present, is there difference between IBG profiles?
- Play with `PcdMeasuredBootHashMask`.
- Perform sealing and unsleaing using tpm2-tools.
- Perform attestation of the TPM quote.
- Add measurement call to SBL boot.

Production approach

- Provisioning
  - Record EK public key (and EK cert if present) for future TPM authentication
  - Create and persist an Attestation Key (AK); enroll AK cert with your Attestation CA.
  - Generate device ID.
  - Consider pre-seal some secret and additional authentication policy to PolicyPCR
- Remote attestation server setup - e.g. Keylime
  - All components build in reproducible manner with attestation in mind (SALSA/in-toto) and result artifacts should feed attestation server.
- Recovery
  - Consider safe mode where dedicated diagnostics OS can be boot and potentially recover (only when additional authentication pass).
- Testing
  - Verify all possible boot paths and firmware state changes.

Conclusion

- Slim Bootloader implements measured boot in very minimalistic way (just PCR0 and PCR1 are extended).
- Slim Bootloader measured boot seem to be more robust than typical IBV implementation.
- Slim Bootloader consist all infrastructure to leverage
- We should know how measured boot works in Slim Bootlaoder in details and what we can do with it.