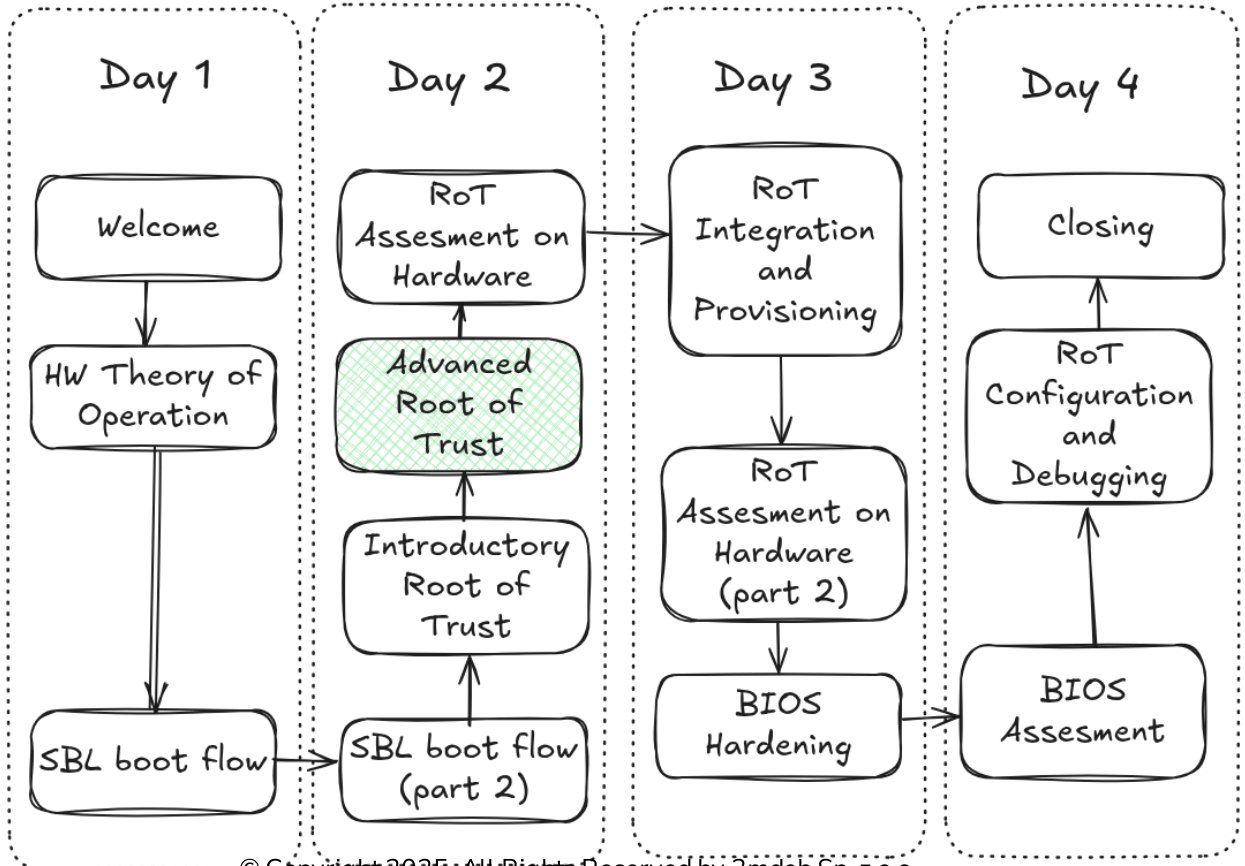
DS09SBL: Dasharo TrustRoot Training

Root of Trust and Chain of Trust Technologies (part 2)

Advanced Root of Trust

Where we are in the course



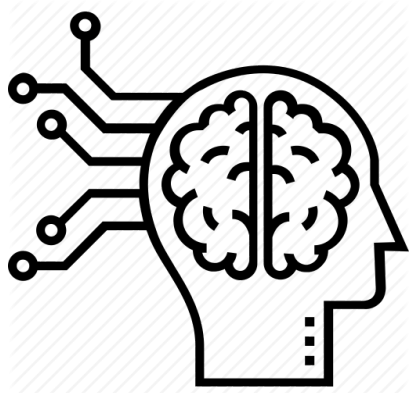
Goals of the Presentation

In this presentation, we aim to:

- Understand difference between Static and Dynamic Root of Trust
- Discuss Example implementations of S-RTM and D-RTM
- Explain role of Intel FIT and ACM for Intel Boot Guard and TXT

S-RTM vs D-RTM ”holy war”

- *Static Root of Trust for Measurement (S-RTM)*
  - requires platform reset to establish a trusted state
  - hard to update (re-evaluation of PCR values)
  - vendor-specific (NXP HAB, Intel Secure Boot/Boot Guard, AMD HVB) requires proprietary tools and NDAs
  - well-established in the IBV environment
  - there are open implementations based on coreboot and vboot (Chromebook, PC Engines)
- *Dynamic Root of Trust for Measurement (D-RTM)*
  - works even in a compromised environment
  - avoid platform reboot to establish a secure state
  - avoid the problem with a secure re-evaluation of PCR values
  - small enough to pass common criteria certification
  - open implementation coming to GRUB ad Linux kernel



- Intel Trusted Execution Technology (TXT)
  - uses binary blobs: ACM BIOS and ACM SINIT modules,
  - leverages `GETSEC[SENTER]` instruction and other leaf functions of it,
- AMD Secure Startup
  - it is possible to create a fully open-source implementation,
  - leverages `SKINIT` instruction introduced with the AMD-V extension,
- Qualcomm - Supposedly comply with Arm D-RTM specification
- Microsoft provides recommendations for their Secured Core PC for Intel, AMD, and Qualcomm SoC.

Quiz

Quiz

What does it mean that RoT is static?

Quiz

What does it mean that RoT is static?

- It performs its assessment only at a specific/static point in time

Quiz

What does it mean that RoT is static?

- It performs its assessment only at a specific/static point in time

What does it mean that RoT is dynamic?

Quiz

What does it mean that RoT is static?

- It performs its assessment only at a specific/static point in time

What does it mean that RoT is dynamic?

- It performs its assessment only at an arbitrary point in time

Quiz

What does it mean that RoT is static?

- It performs its assessment only at a specific/static point in time

What does it mean that RoT is dynamic?

- It performs its assessment only at an arbitrary point in time

How is D-RTM typically implemented?

Quiz

What does it mean that RoT is static?

- It performs its assessment only at a specific/static point in time

What does it mean that RoT is dynamic?

- It performs its assessment only at an arbitrary point in time

How is D-RTM typically implemented?

- By dedicated CPU instruction

Quiz

What does it mean that RoT is static?

- It performs its assessment only at a specific/static point in time

What does it mean that RoT is dynamic?

- It performs its assessment only at an arbitrary point in time

How is D-RTM typically implemented?

- By dedicated CPU instruction

What are the 2 main D-RTM threats?

Quiz

What does it mean that RoT is static?

- It performs its assessment only at a specific/static point in time

What does it mean that RoT is dynamic?

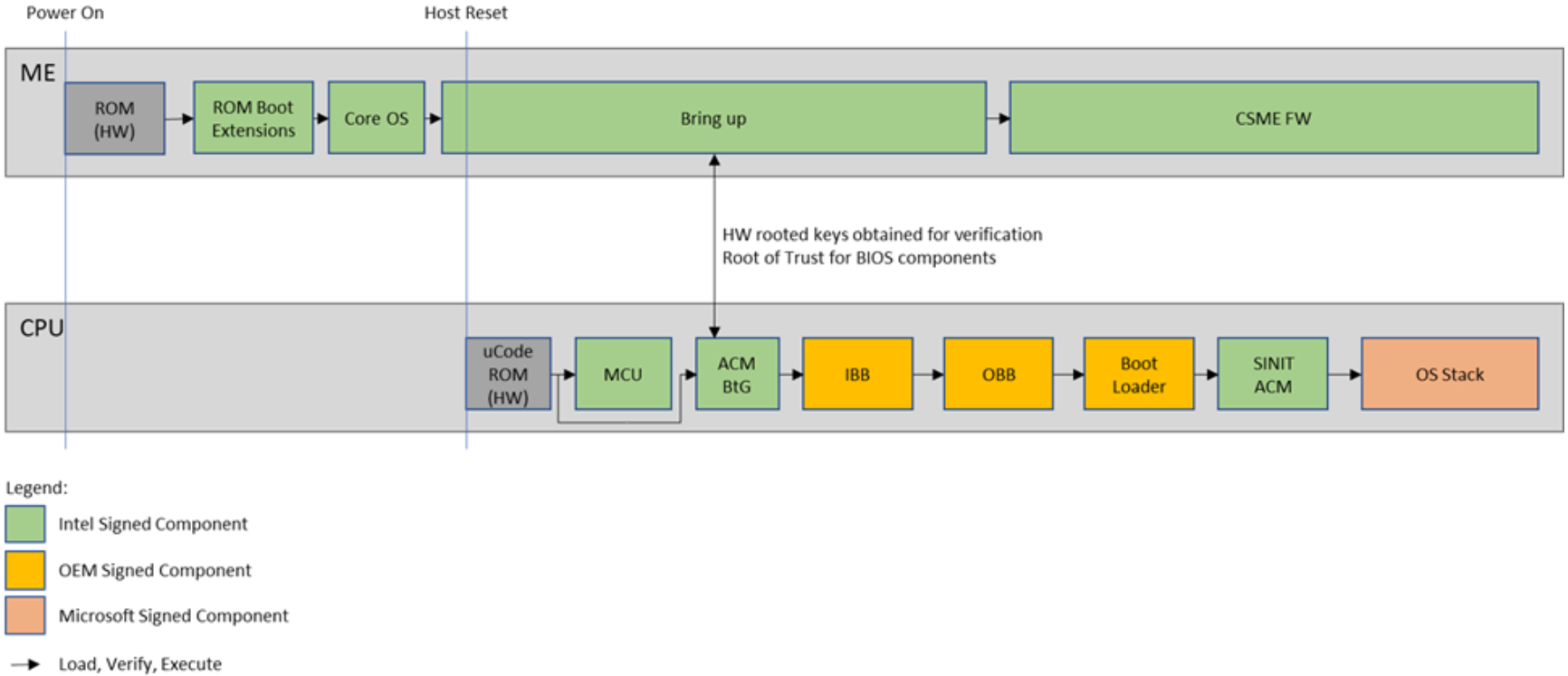
- It performs its assessment only at an arbitrary point in time

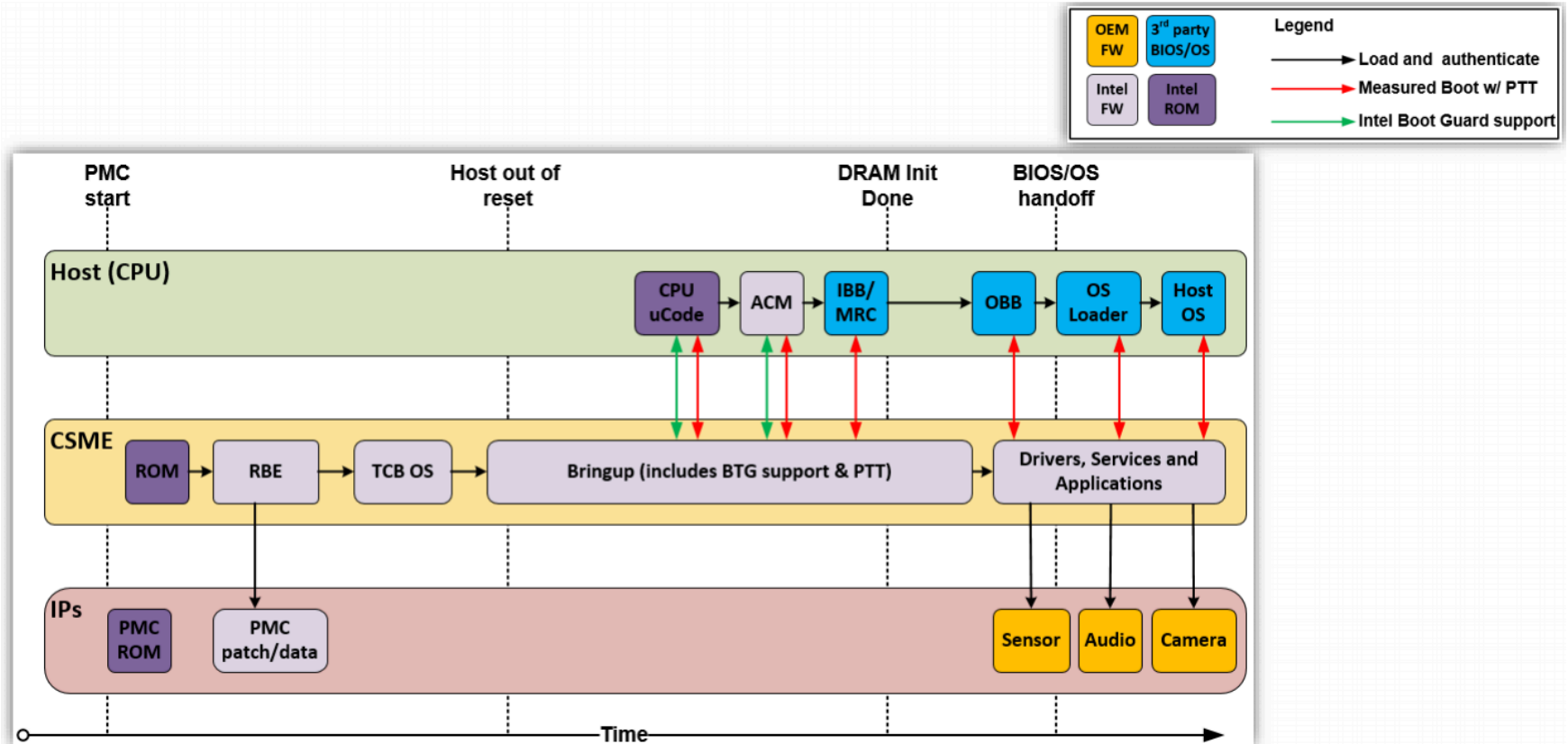
How is D-RTM typically implemented?

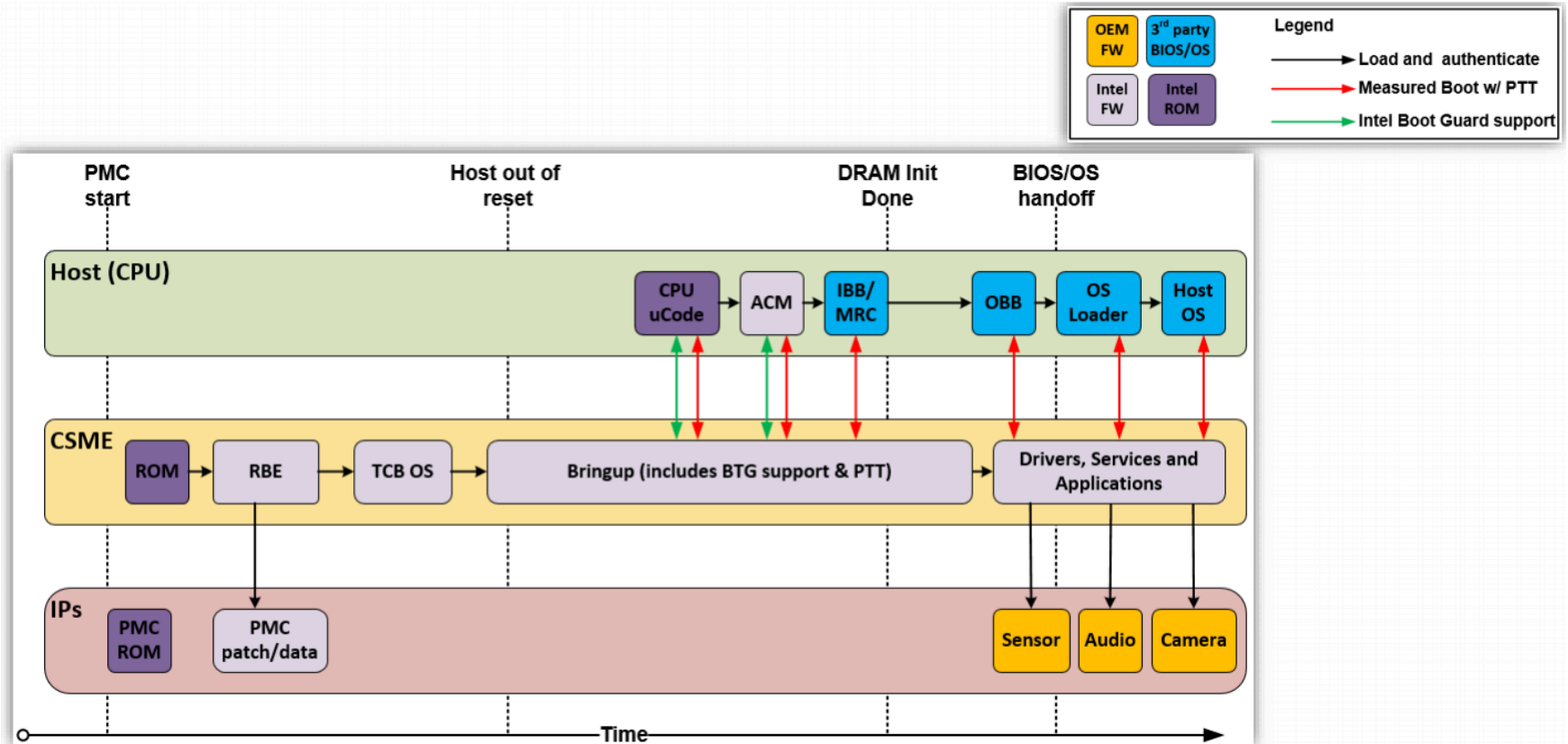
- By dedicated CPU instruction

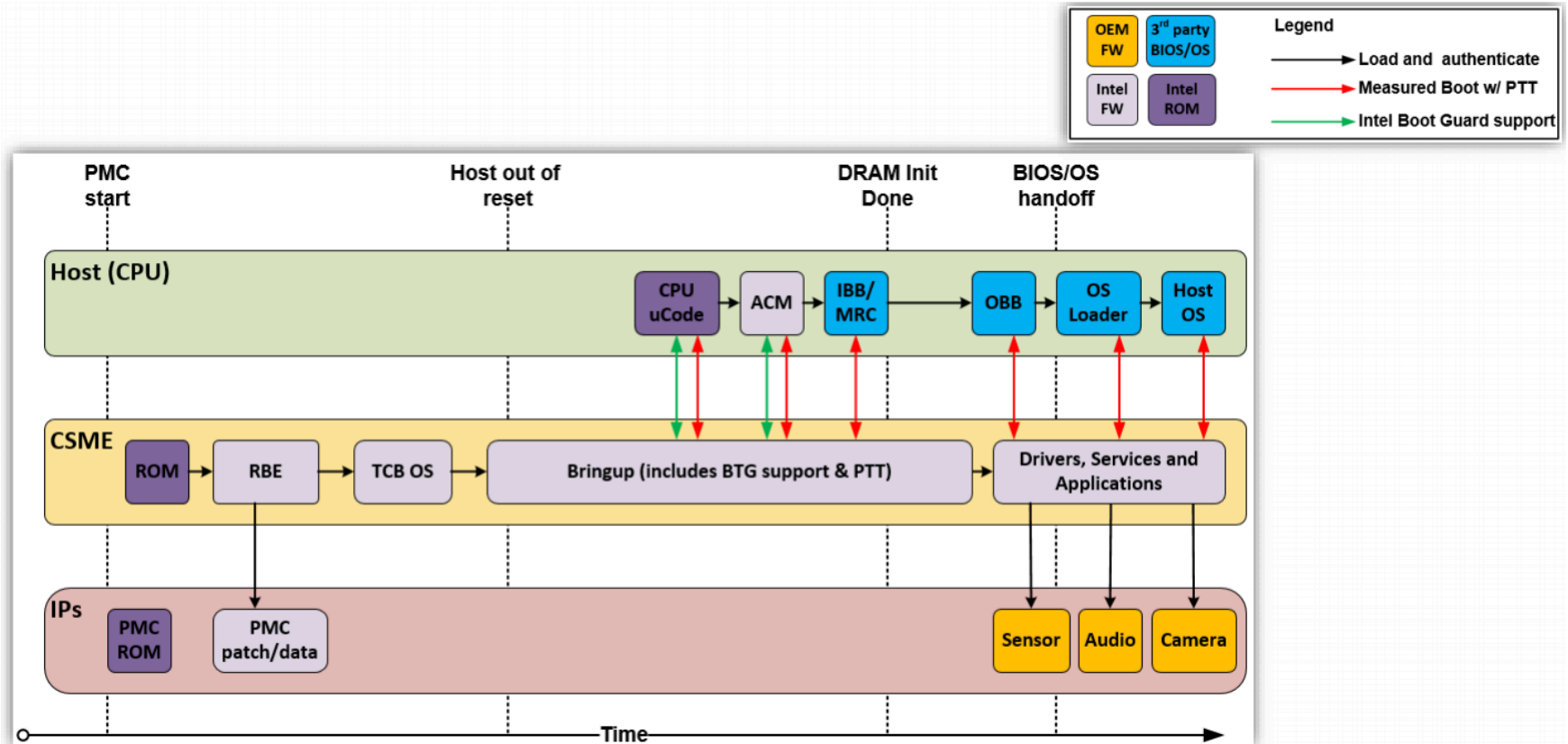
What are the 2 main D-RTM threats?

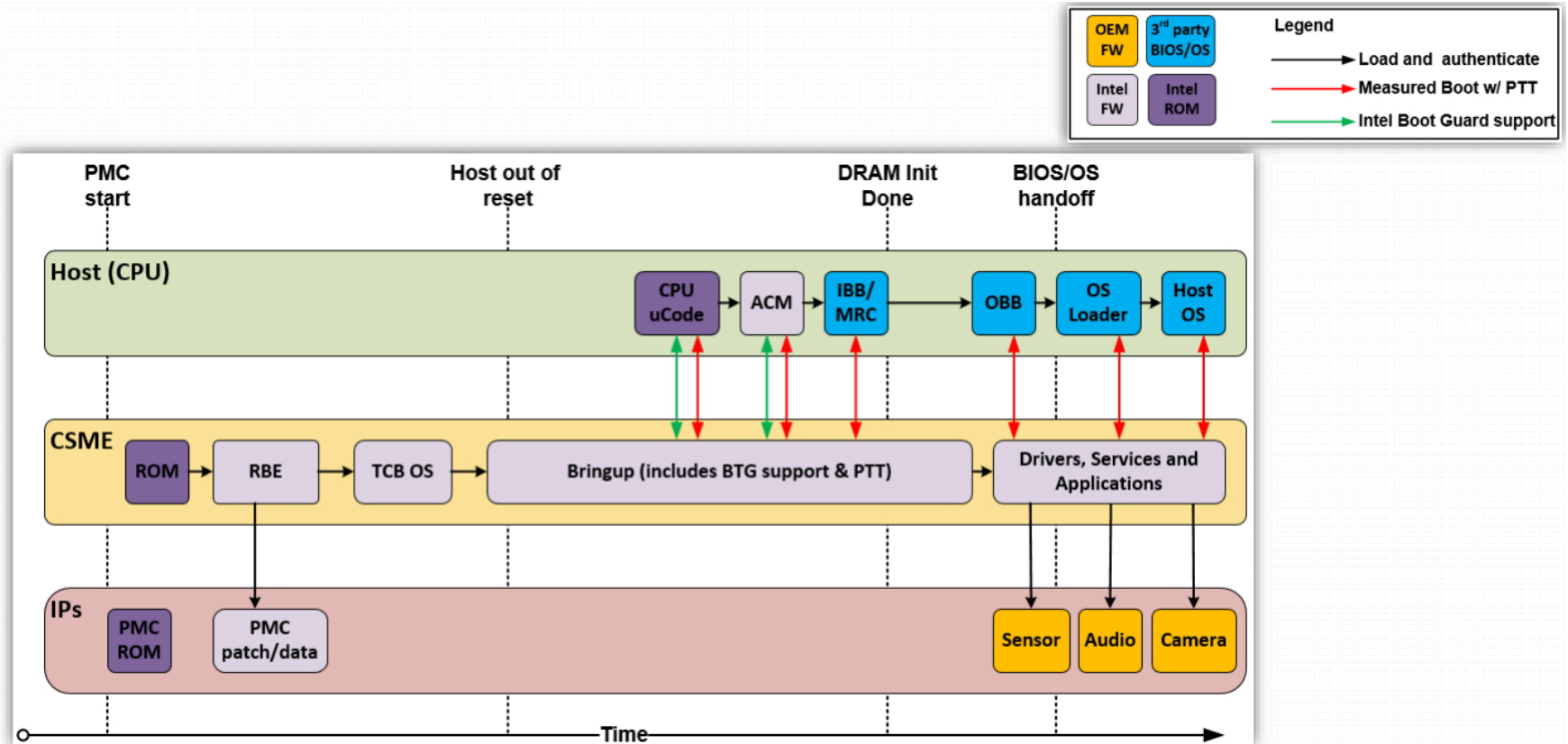
- SMI attacks
- DMA attacks

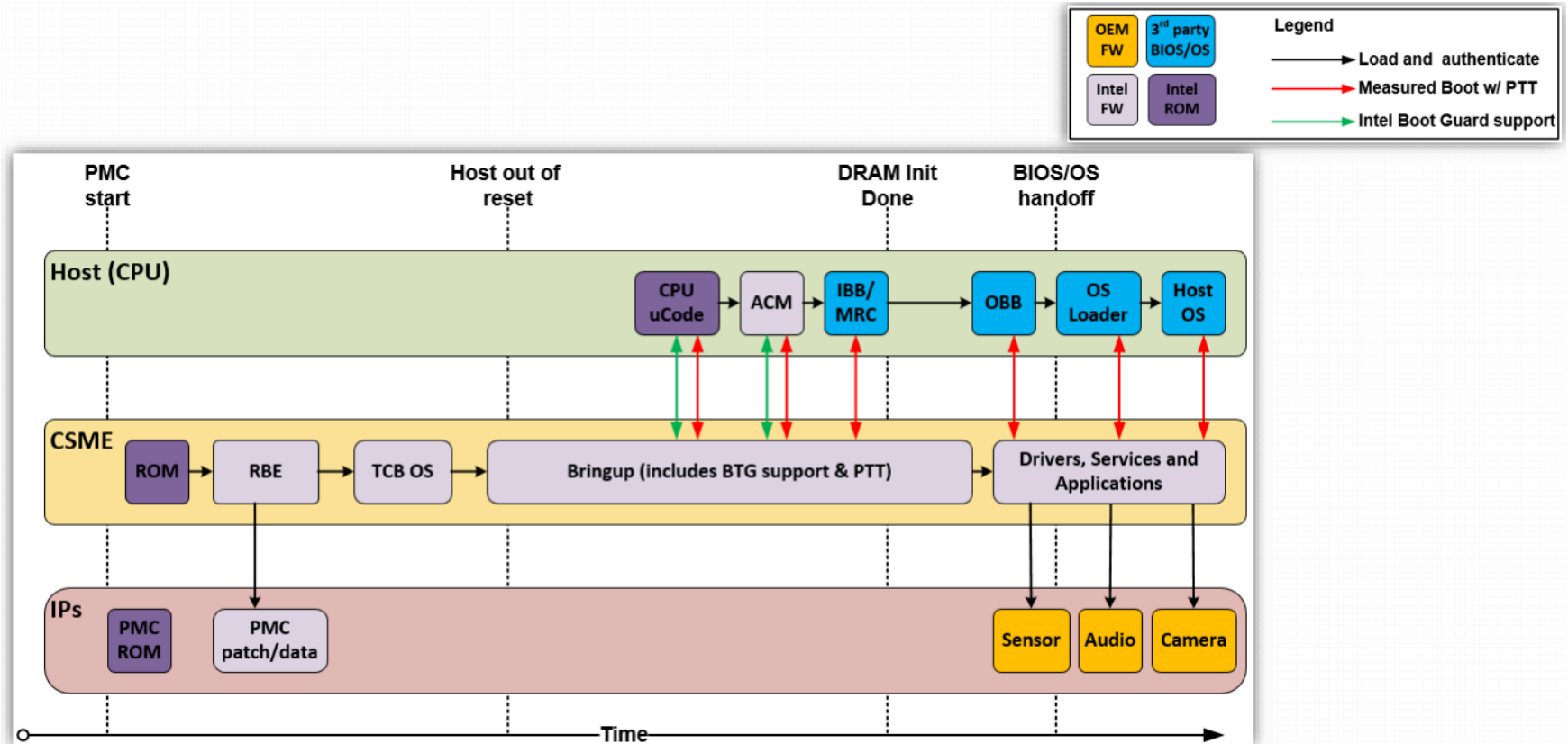












```
U0000: 00626803f200               tmp15:= MOVEFROMCREG_DSZ64(CORE_CR_CUR_UIP)
U0001: 000801030008               tmp0:= ZEROEXT_DSZ32(0×00000001)
           018e5e40               SEQW GOTO U0e5e
──────────────────────────────────────────────────────────────────────────────
U0002: 004800013000               tmp7:= ZEROEXT_DSZ64(0×00000000)

U0004: 05b900013000               mm7:= unk_5b9(0×00000000)
U0005: 000a01000200               TESTUSTATE(UCODE, UST_MSLOOPCTR_NONZERO)
           0b000240               ? SEQW GOTO U0002
U0006: 014800000000     SYNCWAIT→ URET(0×00)
──────────────────────────────────────────────────────────────────────────────

U0008: 000c6c97e208               tmp14:= SAVEUIP(0×01, U056c)
           01890900               SEQW GOTO U0909
```

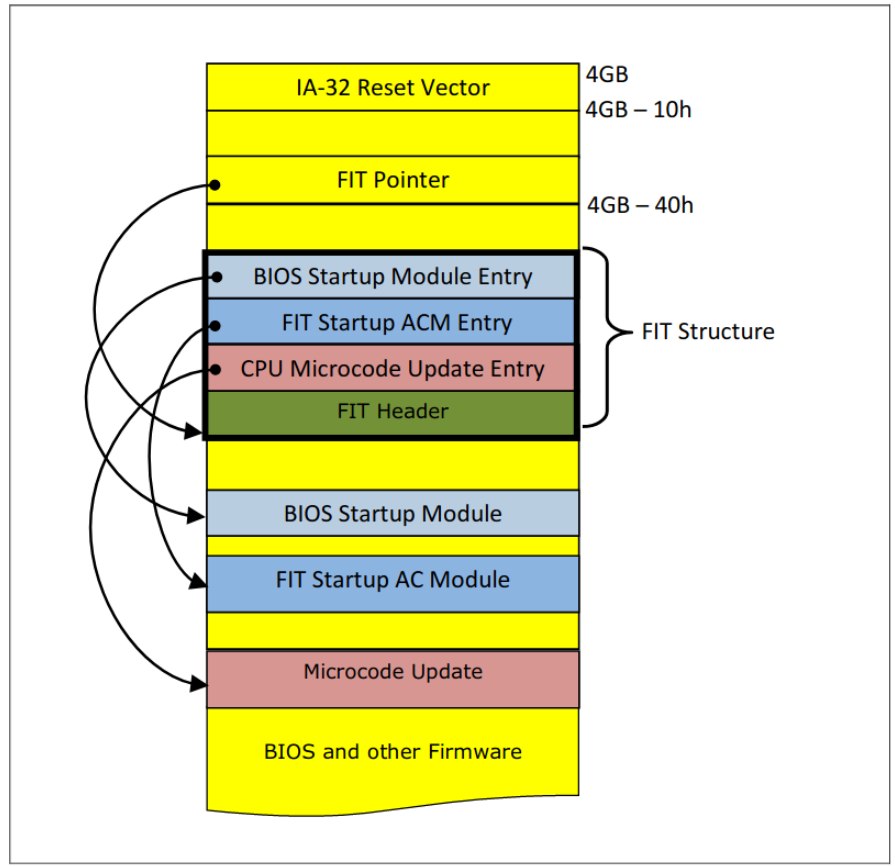
- Microcode is probably the lowest level machine code.
- Implements the programmer-visible instruction set architecture.
- Intel microcode is available on GitHub.
- Signed and encrypted by Intel to prevent unauthorized microcode updates.
- The key is burned into the CPUs.
- Used to authenticate ACMs.

source: Disassembled microcode source Chip Red Pill Team

FIT table

- The Firmware Interface Table (FIT) is a data structure inside the BIOS binary that describes certain BIOS attributes.
- The absolute address of the beginning of the table is located at a physical address of 4GB - 40h (64 bytes from the top of the flash).
- It is processed by CPU **before** the first BIOS instruction at the reset vector is executed.
- Contains pointers to microcode updates and ACMs (Authenticated Code Modules) to be executed, etc.
- FIT is one of the structures used to establish a Root of Trust in Intel-based systems.
- It is documented in `599500_FW_Interface_Table_BIOS_Spec_Rev1p5.pdf`:
  - "The BIOS flash must include a Firmware Interface Table (FIT) with Type 0 (FIT Header) and Type 1 (Microcode Update) entries."
  - "A microcode update must exist for every processor stepping supported by the platform."

Figure 2-1. FIT Layout in Flash / ROM

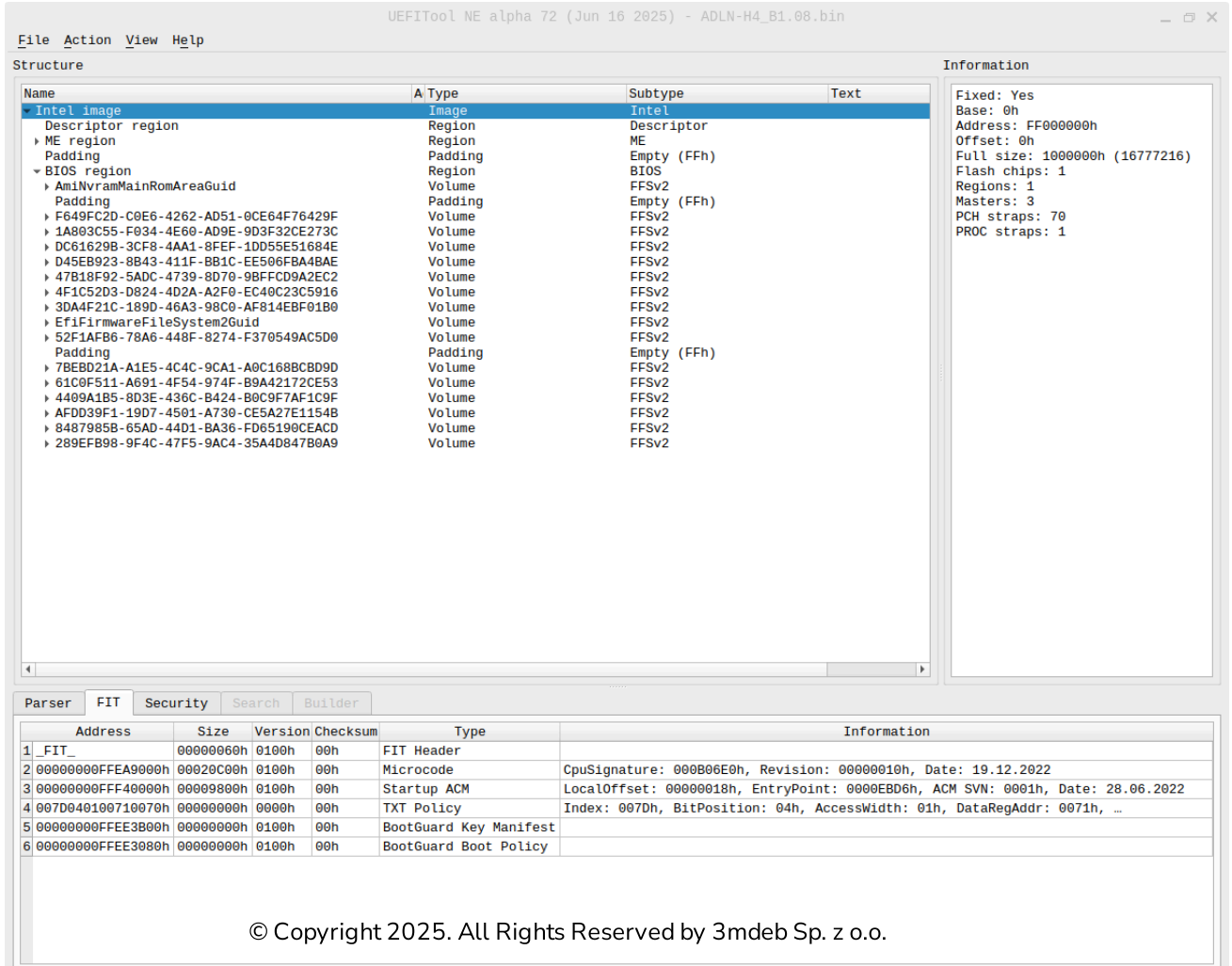


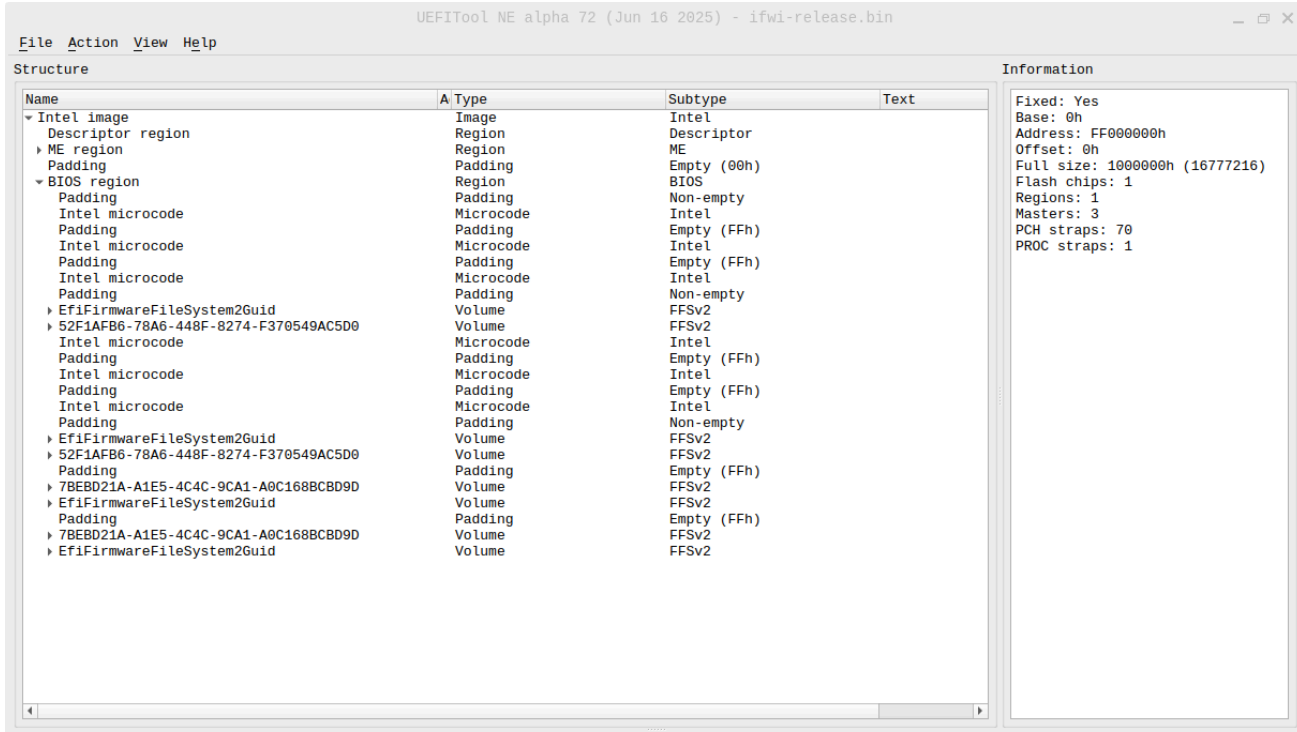
Practice #301 Exercise #0

Find ucode in the AMI BIOS and Slim Bootloader image using UEFI Tool or any other tool of your choice.

The UEFI Tool should be started graphically, for example via RDP.

- AMI BIOS binary is in `$HOME/training_materials/hardware/odroid/bios`





Practice #301 Exercise #1

Find a FIT table in the image and decode all entry types it contains.

- UEFI Tool is in your `$HOME/training_materials/uefi` directory.
- AMI BIOS binary is in `$HOME/training_materials/hardware/odroid-h4/bios`
- Dasharo (Slim Bootloader+UEFI) binary should be in `$HOME/training_materials/src/slimbootloader/Output`
- FIT Entry Types can be found in documentation in Table 4-2 (https://edc.intel.com/content/www/br/pt/design/products-and-solutions/software-and-services/firmware-and-bios/firmware-interface-table/1.6/firmware-interface-table/).

Parser | FIT | Security | Search | Builder

	Address	Size	Version	Checksum	Type	Information
1	_FIT_	00000070h	0100h	00h	FIT Header	
2	00000000FFDD0400h	00017C00h	0100h	00h	Microcode	CpuSignature: 000506E3h, Revision: 000000B2h, Date: 01.02.2017
3	00000000FFDE8000h	00017800h	0100h	00h	Microcode	CpuSignature: 000906E9h, Revision: 00000048h, Date: 15.11.2016
4	00000000FFDFF800h	00017800h	0100h	00h	Microcode	CpuSignature: 000506E8h, Revision: 00000034h, Date: 10.07.2016
5	00000000FFFD0000h	00008000h	0100h	00h	BIOS ACM	LocalOffset: 00000018h, EntryPoint: 00003BB1h, ACM SVN: 0000h, Date: 28.08.2016
6	00000000FFFD9100h	00000241h	0100h	00h	BootGuard Key Manifest	
7	00000000FFFD8080h	000002DFh	0100h	00h	BootGuard Boot Policy	

- Signed by Intel binary blobs to initialize
  - Intel Boot Guard - technology used as HRTM, RTD, RTV and RTM.
  - BIOS Guard - technology used as RTU.
  - TXT - technology used for DRTM.
  - And probably other future technologies.

Intel Authenticated Code Module

- ACMs are authenticated by microcode and a key burned in CPU.

- ACMs are authenticated by microcode and a key burned in CPU.
- TXT ACMs
  - BIOS ACMs
  - SINIT ACMs

- ACMs are authenticated by microcode and a key burned in CPU.
- TXT ACMs
  - BIOS ACMs
  - SINIT ACMs
- BIOS Guard ACMs

- ACMs are authenticated by microcode and a key burned in CPU.
- TXT ACMs
  - BIOS ACMs
  - SINIT ACMs
- BIOS Guard ACMs
- Boot Guard ACMs

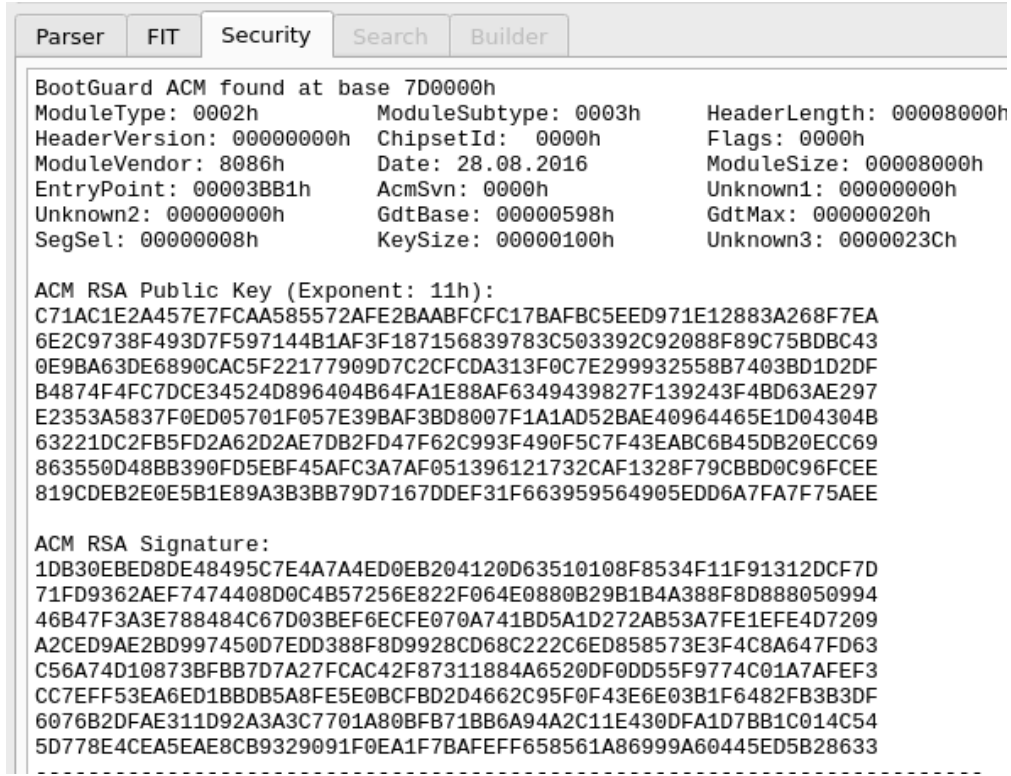
- ACMs are authenticated by microcode and a key burned in CPU.
- TXT ACMs
  - BIOS ACMs
  - SINIT ACMs
- BIOS Guard ACMs
- Boot Guard ACMs
- Intel TDX ACMs

- Startup ACMs (S-ACM) (FIT entry type 0x2)

- Startup ACMs (S-ACM) (FIT entry type 0x2)
- Diagnostics ACMs (FIT entry type 0x3)

- Startup ACMs (S-ACM) (FIT entry type 0x2)
- Diagnostics ACMs (FIT entry type 0x3)
- Runtime ACMs (non-official nomenclature)

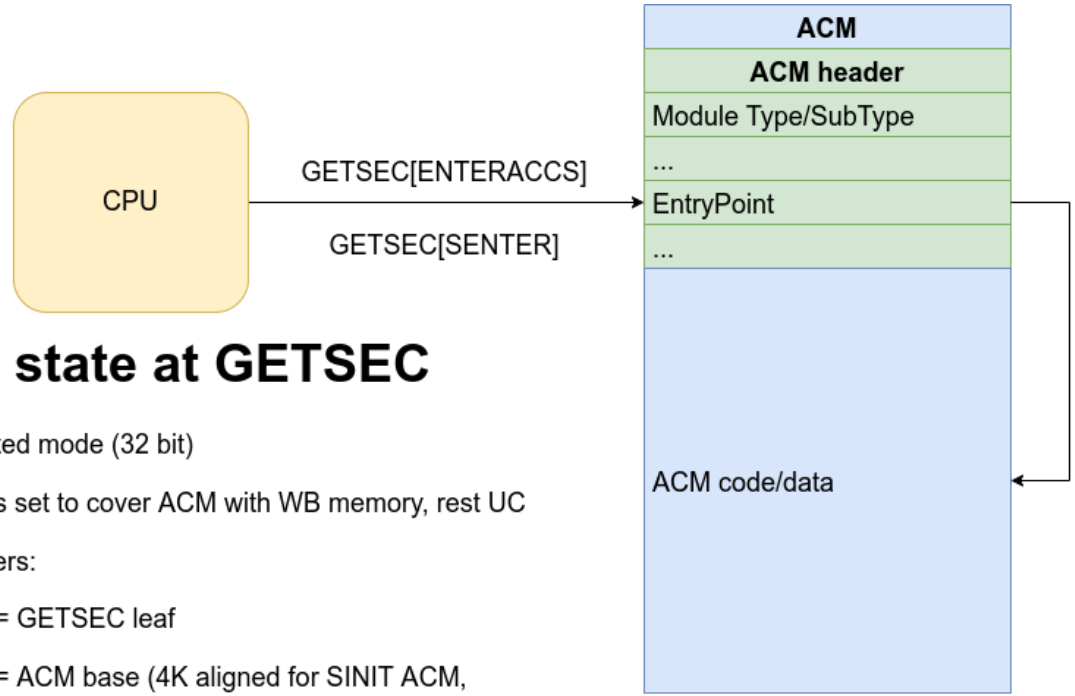
- SlimBootloader: BuildLoader
- ACM Type 3
- Get Started with Intel® Time Coordinated Computing Tools (Intel® TCC Tools) 2022.1 for Slim Bootloader
- Intel TDX Documanation
- Intel TDX Interface Specification



ACM structure definition may be found in Appendix A of Intel Trusted Execution Measured Launch Environment Developer's Guide

Intel SMX

- Safer Mode Extension provide a programming interface to establish measured environment (Intel TXT)
  - described in Intel SDM chapter 6
  - key technology to implement DRTM for Intel platforms
- Includes:
  - Measured Launched Environment (MLE), which may be based on measured VMM (MVMM)
  - secure storage and location for measurement
  - VMM protection against modification
- SMX functionality is provided by `GETSEC` instruction
- BIOS is responsible for loading ACM and calling `GETSEC[ENTERACCS]`
  - In contrast to Boot Guard, where the ACM is loaded and `GETSEC[ENTERACCS]` is executed automatically by the Boot ROM
- This extension is not available in all processors and requires compatible chipset be fully functional.



CPU state at GETSEC

1. Protected mode (32 bit)

2. MTRRs set to cover ACM with WB memory, rest UC

3. Registers:

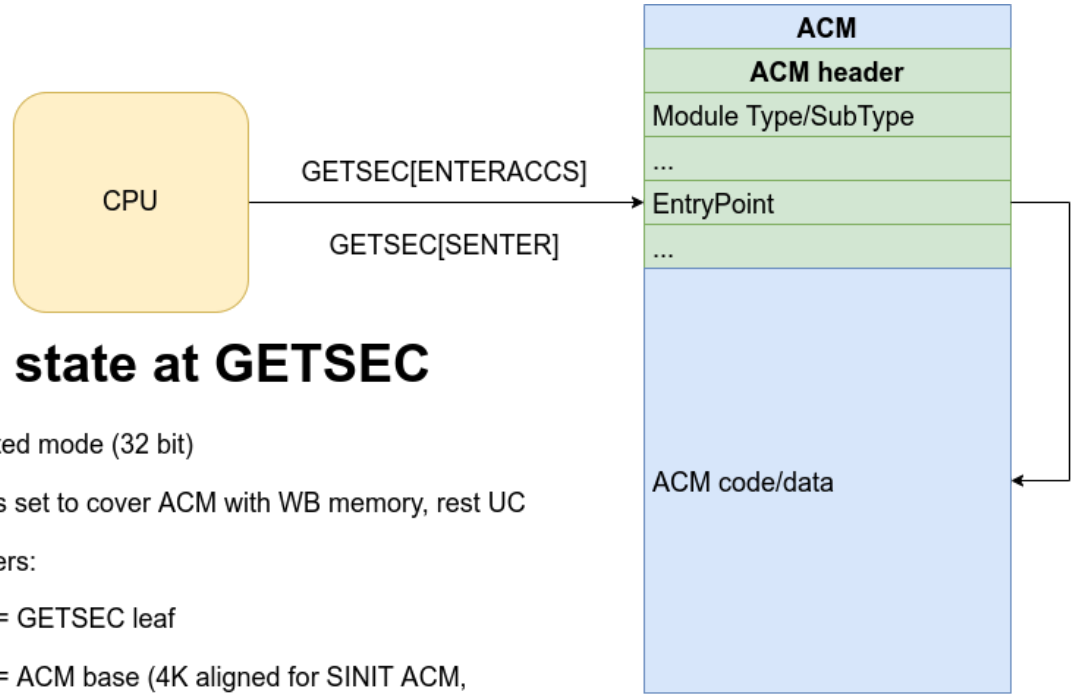
   - EAX = GETSEC leaf

   - EBX = ACM base (4K aligned for SINIT ACM,
     64K/128K/256K for BIOS ACM
     platform-specific)

   - ECX = ACM size

   - EDX, EDI = 0

   - ESI = BIOS ACM function to call, else 0



CPU state at GETSEC

1. Protected mode (32 bit)

2. MTRRs set to cover ACM with WB memory, rest UC

3. Registers:

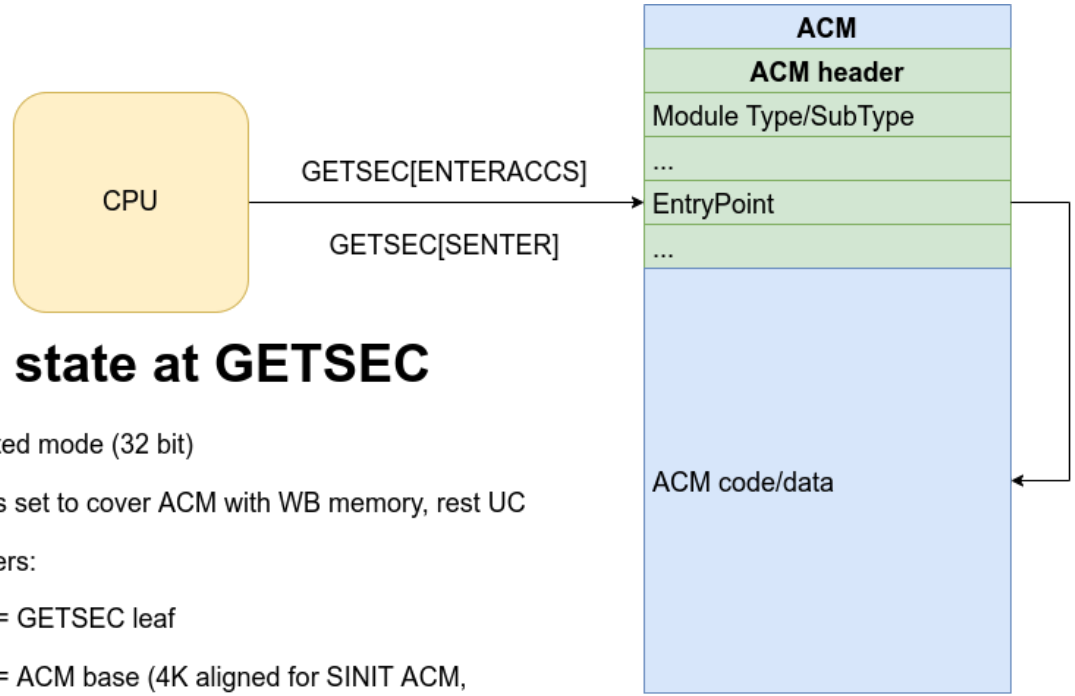
   - EAX = GETSEC leaf

   - EBX = ACM base (4K aligned for SINIT ACM,
     64K/128K/256K for BIOS ACM
     platform-specific)

   - ECX = ACM size

   - EDX, EDI = 0

   - ESI = BIOS ACM function to call, else 0



CPU state at GETSEC

1. Protected mode (32 bit)

2. MTRRs set to cover ACM with WB memory, rest UC

3. Registers:

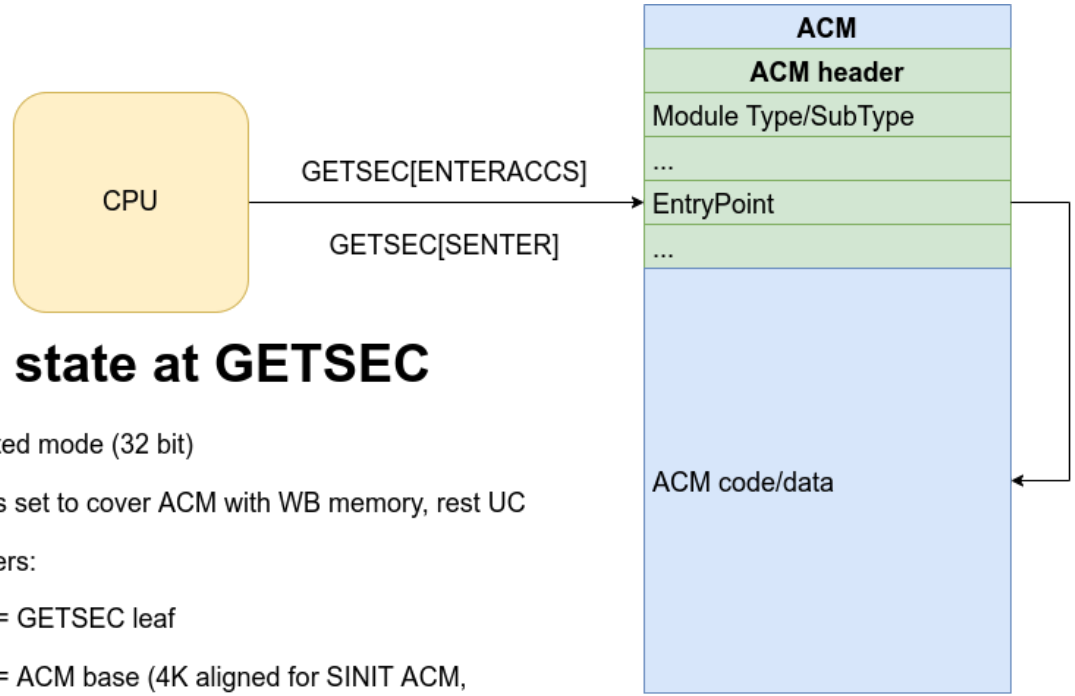
   - EAX = GETSEC leaf

   - EBX = ACM base (4K aligned for SINIT ACM,
     64K/128K/256K for BIOS ACM
     platform-specific)

   - ECX = ACM size

   - EDX, EDI = 0

   - ESI = BIOS ACM function to call, else 0



CPU state at GETSEC

1. Protected mode (32 bit)

2. MTRRs set to cover ACM with WB memory, rest UC

3. Registers:

   - EAX = GETSEC leaf

   - EBX = ACM base (4K aligned for SINIT ACM,
     64K/128K/256K for BIOS ACM
     platform-specific)

   - ECX = ACM size

   - EDX, EDI = 0

   - ESI = BIOS ACM function to call, else 0

Quiz

Quiz

What is Authenticated Code Module?

Quiz

What is Authenticated Code Module?

- Signed by Intel binary blob to initialize Boot Guard, TXT, BIOS Guard, TDX and FuSa.

Quiz

What is Authenticated Code Module?

- Signed by Intel binary blob to initialize Boot Guard, TXT, BIOS Guard, TDX and FuSa.

What types of ACMs form Root of Trust?

Quiz

What is Authenticated Code Module?

- Signed by Intel binary blob to initialize Boot Guard, TXT, BIOS Guard, TDX and FuSa.

What types of ACMs form Root of Trust?

- BIOS ACM - Static Root of Trust
- SINIT ACM - Dynamic Root of Trust
- TDX ACM - Root of Trust for Confidentiality

Quiz

What is Authenticated Code Module?

- Signed by Intel binary blob to initialize Boot Guard, TXT, BIOS Guard, TDX and FuSa.

What types of ACMs form Root of Trust?

- BIOS ACM - Static Root of Trust
- SINIT ACM - Dynamic Root of Trust
- TDX ACM - Root of Trust for Confidentiality

What instruction is used to execute BIOS ACM?

Quiz

What is Authenticated Code Module?

- Signed by Intel binary blob to initialize Boot Guard, TXT, BIOS Guard, TDX and FuSa.

What types of ACMs form Root of Trust?

- BIOS ACM - Static Root of Trust
- SINIT ACM - Dynamic Root of Trust
- TDX ACM - Root of Trust for Confidentiality

What instruction is used to execute BIOS ACM?

- `GETSEC[ENTERACCS]`

Leaks Effect

- Recent leaks of cryptographic materials seem to do a lot of good job for quality and transparency of documentation.

> **Quote**
>
> This technical article is intended to address an issue discovered in some Original Equipment Manufacturers (OEM) or Original Design Manufacturers (ODM) production systems where system firmware (FW) or an Integrated Firmware Image (IFWI) **was found to contain one or more preproduction or example test keys**. The private keys for these preproduction or test keys were often included in system Basic Input Output System (BIOS) development information provided to OEMs or ODMs for their use in platform development and testing and were not intended to be used in production systems or environments.

source: Introduction to Key Usage in Integrated Firmware Images

- Limitation

> **Quote**
>
> This document’s information is generally limited to products based on 13th Generation Intel® Core™ Processor platforms and 3rd Generation Intel® Xeon® Scalable Processor platforms, or earlier. Guidance for manifesting and signing components for future products is subject to change.

source: Introduction to Key Usage in Integrated Firmware Images

> **Quote**
>
> Platform manufacturers use field programmable fuses (FPFs) to provision the boot policy to be used on the system. They provide the foundational mechanism for verifying the integrity of the KM and BPM.

source: *Introduction to Key Usage in Integrated Firmware Images*

- FPF are persistently stored in Platform Controller Hub (PCH) in case of dual chipset design (CPU+PCH) and in SoC for mobile/embedded systems - this is our key store backed by some fusing technology.
- Once FPFs are provisioned a hardware lock is closed and those cannot be changed for the life of the platform.
- Establishing final lock is referred to as End of Manufacturing (EOM).
- FPFs are not accessible directly and can be controlled only through ME/CSE/CSME, which is inside PCH or SoC.

Quiz

Quiz

Where Field Programmable Fuses are located in modern Intel platform?

Quiz

Where Field Programmable Fuses are located in modern Intel platform?

- In case of single chip platform fuses are in SoC.
- In case of dual-chip platform (CPU+PCH) fuses are in PCH.

Quiz

Where Field Programmable Fuses are located in modern Intel platform?

- In case of single chip platform fuses are in SoC.
- In case of dual-chip platform (CPU+PCH) fuses are in PCH.

What effect on Silicon Vendor may have security vulnerabilities?

Quiz

Where Field Programmable Fuses are located in modern Intel platform?

- In case of single chip platform fuses are in SoC.
- In case of dual-chip platform (CPU+PCH) fuses are in PCH.

What effect on Silicon Vendor may have security vulnerabilities?

- They have to explain impact of vulnerability, what typically means revealing NDA design of the internals of the platform design.
- This is needed to explain to stakeholders that required mitigation was applied.
- Ecosystem gain transparency and improve auditability on the cost of temporary instability of the global computer ecosystem.

Quiz

Where Field Programmable Fuses are located in modern Intel platform?

- In case of single chip platform fuses are in SoC.
- In case of dual-chip platform (CPU+PCH) fuses are in PCH.

What effect on Silicon Vendor may have security vulnerabilities?

- They have to explain impact of vulnerability, what typically means revealing NDA design of the internals of the platform design.
- This is needed to explain to stakeholders that required mitigation was applied.
- Ecosystem gain transparency and improve auditability on the cost of temporary instability of the global computer ecosystem.

What effect on Silicon Vendor may have key and source code leaks?

Quiz

Where Field Programmable Fuses are located in modern Intel platform?

- In case of single chip platform fuses are in SoC.
- In case of dual-chip platform (CPU+PCH) fuses are in PCH.

What effect on Silicon Vendor may have security vulnerabilities?

- They have to explain impact of vulnerability, what typically means revealing NDA design of the internals of the platform design.
- This is needed to explain to stakeholders that required mitigation was applied.
- Ecosystem gain transparency and improve auditability on the cost of temporary instability of the global computer ecosystem.

What effect on Silicon Vendor may have key and source code leaks?

Conclusion

In this lecture, we learned about the following:

- Even the best SRTM requires high quality initial protection and SRTM boot chains as well as protecting it RoT are complex.
- Root of Trust does not have to be static. Dynamic RoT may bring complexity as well as opportunity to improve security posture of the platform and system.
- Reset vector is not beginning of modern x86 boot process.
- Where is Intel Boot Guard, what role ACM and uCode have in establishing platform security.
- What is assumed responsibility of various parities in the process according to Intel.

This information will provide us with a solid foundation for:

- Assessment of Intel Boot Guard state.
- Guiding future decision about system security.

Q&A