Shai Hasarfaty   **Principal Security Research Engineer, Intel Corp.**
Yanai Moyal   **Security Researcher, Intel Corp.**
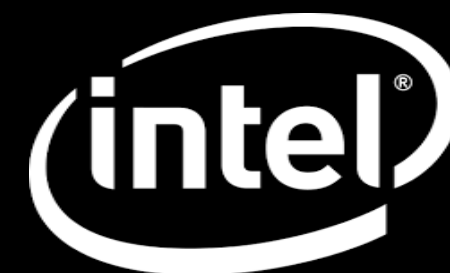
Behind the Scenes of Intel Security and Manageability Engine

#BHUSA @BLACK HAT EVENTS

Intel provides these materials as-is, with no express or implied warranties.

All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice.

Intel, processors, chipsets, and desktop boards may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No product or component can be absolutely secure. Check with your system manufacturer or retailer or learn more at **https://www.intel.com**.

Some results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Intel and the Intel logo are trademarks of Intel Corporation in the United States and other countries.

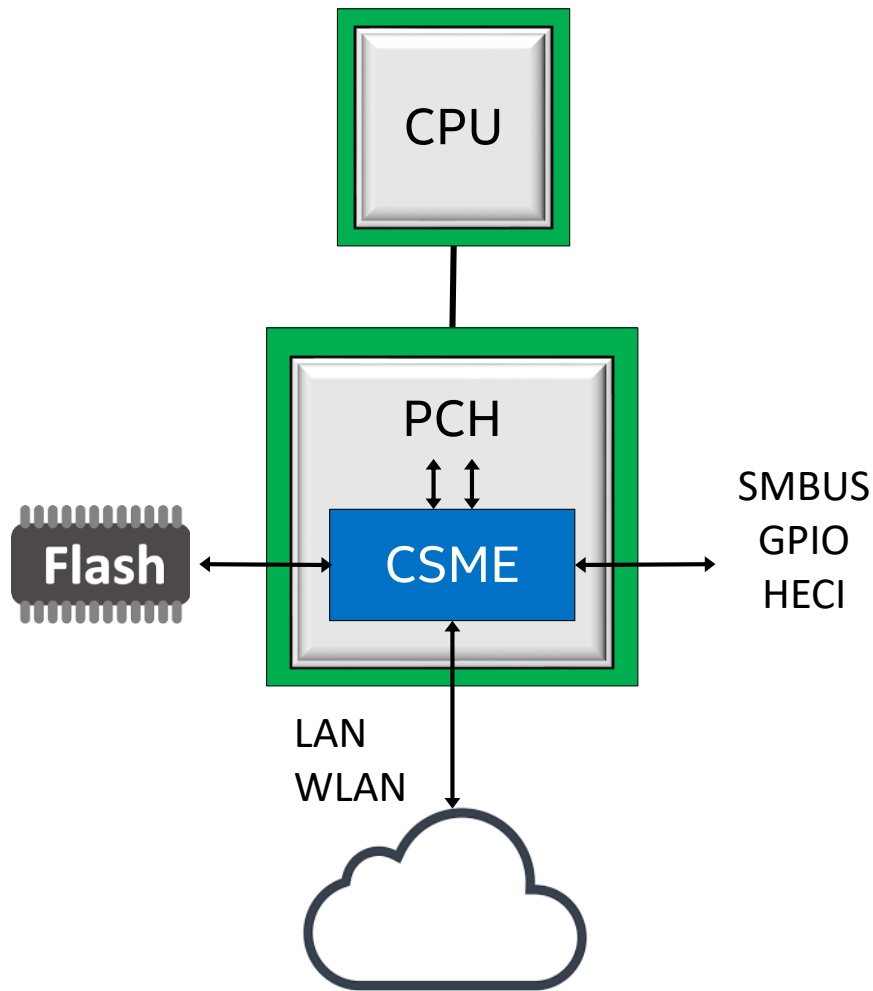*Other names and brands may be claimed as the property of others.

Latest CSME 12 Firmware & Hardware on Intel 8th and 9th Gen Core Processor based platforms (code name Coffee Lake and Whiskey Lake)

- **Architecture & Boot flow**
- OS Security Principles & Internals
- Hardening & Mitigations
- Pre & Post Manufacturing
- Update & Recoverability
- Wrap-up

CSME is an embedded subsystem in Platform Controller Hub (PCH)

• Stands for **C**onverged **S**ecurity & **M**anageability **E**ngine

• Standalone low power Intel processor with dedicated Hardware (HW)

• CSME is Root of Trust of the platform

  • Provides an isolated execution environment protected from host SW running on main CPU

  • Executes CSME Firmware (FW)

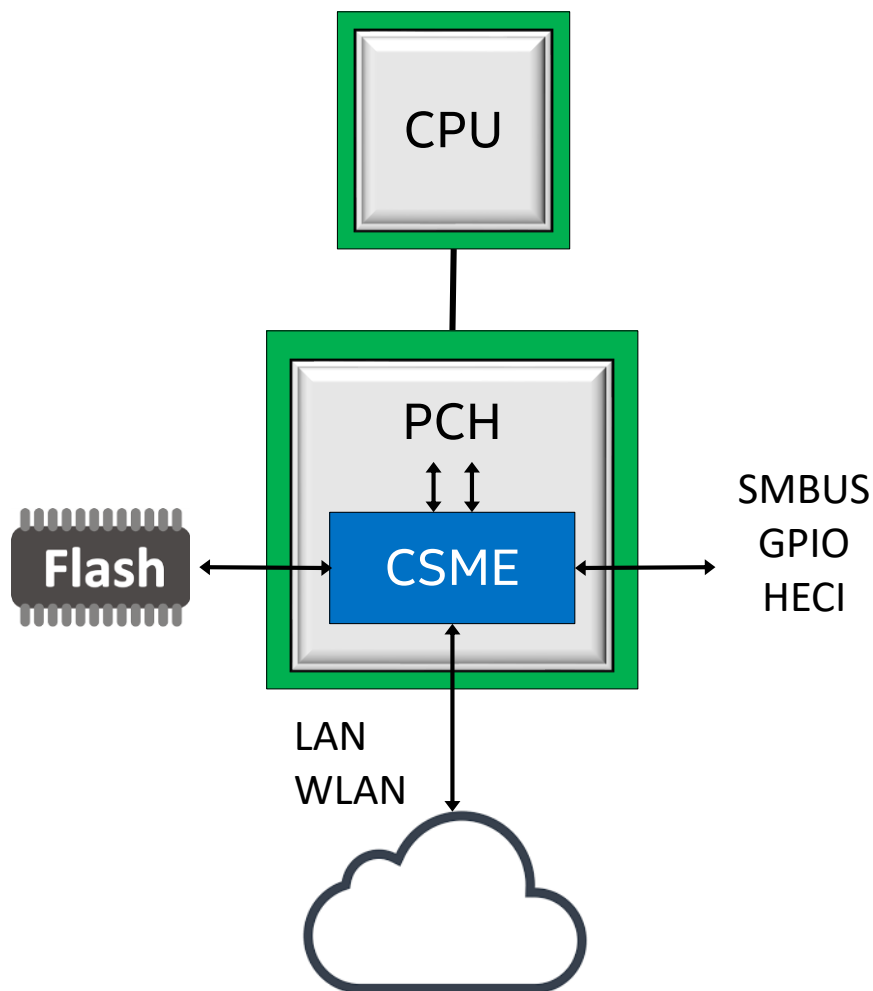**CSME serves 3 main platform roles**

- **Chassis**
  - Secure boot of the platform
  - Overclocking
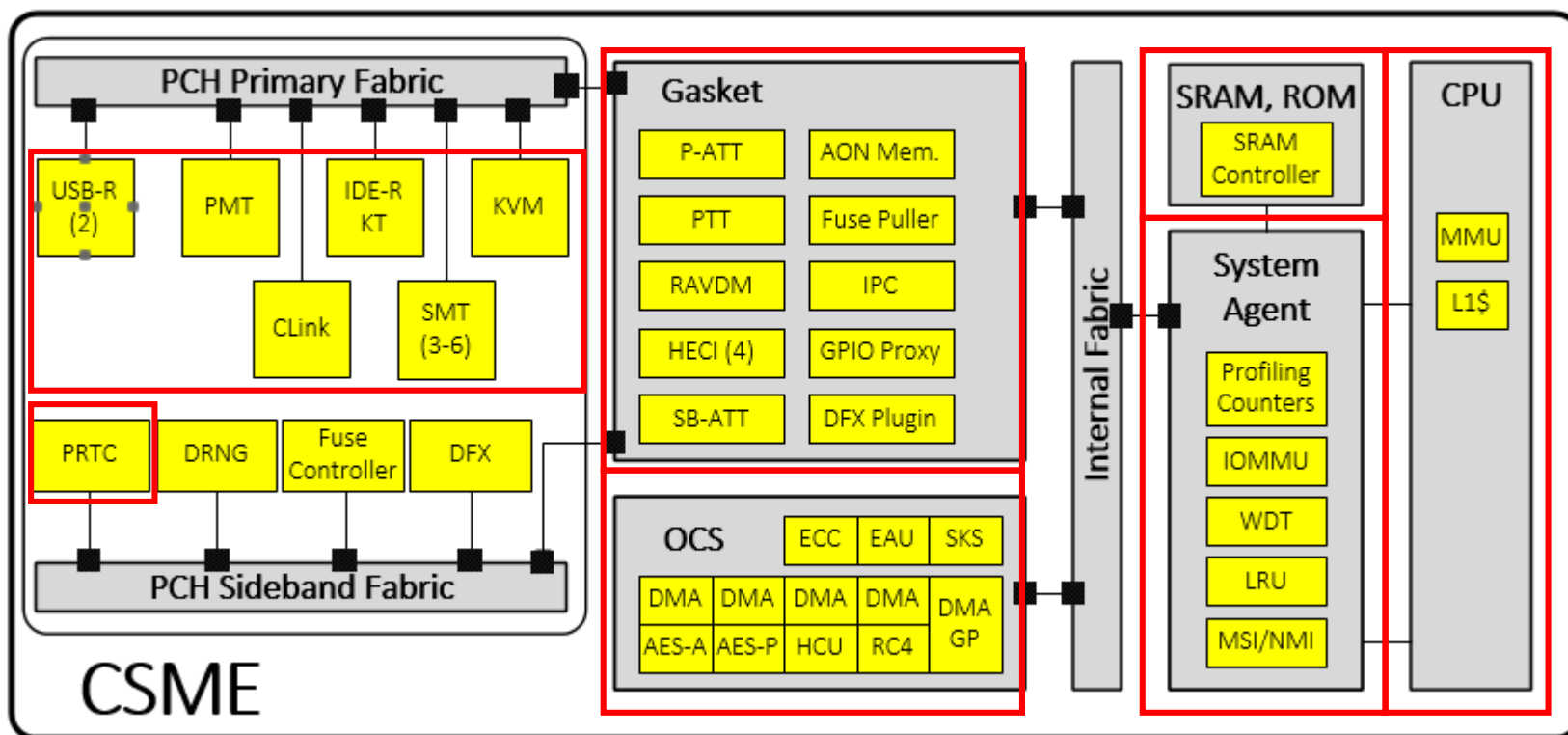 - Micro-code loading into PCH/CPU HW engines
- **Security**
  - Isolated & trusted execution of security services (TPM, DRM, DAL)
- **Manageability**
 - Platform management over out of band network (Intel AMT)

- **CPU:** Intel 32 bits processor (i486) supporting rings, segmentation and MMU for page management

- **SRAM:** Isolated RAM (~1.5 MB) from host

- **ROM:** HW root of trust of CSME Firmware

- **System Agent:** Allows CPU to securely access SRAM and enforce access control to SRAM from internal/external devices by using IOMMU (i.e. control DMA access)

- **OCS (Offload & Cryptography Subsystem):** Crypto HW accelerator with DMA engine and Secure Key Storage (SKS)

- **Gasket:** interface to PCH fabric & CSME IO devices (TPM, HECI etc.)

- **Manageability Devices:** used for manageability and redirection (USB-R, IDE-R, KT, KVM etc.)

- **Protected Real Time Clock:** used for monotonic counters (anti-replay protection) and as protected time
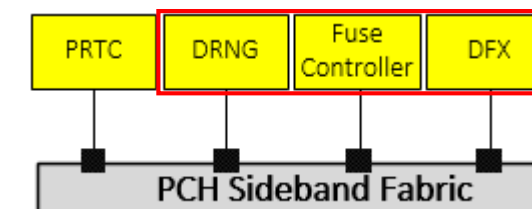
- ## DRNG
  - Generates non-deterministic random numbers
  - Compliant to NIST SP800-90A, B and C

- ## Fuses – 2 types
  - Intel PCH Manufacturing Fuses – set by Intel before shipment to manufacturers
    - CSME configurations: which Intel CSME signing keys enabled, production silicon etc.
    - CSME security keys unique per chip and encrypted using CSME HW key

  - In Field Programmable Fuses (FPF) – set by manufacturers before shipment to end-users
    - Manufacturers' secure settings: public key, Intel Boot Guard policy etc.
    - CSME FW Anti-Rollback Security Version Number (ARB SVN)

- ## DFX (debug)
  - Control CSME & other PCH micro-controllers debug interface (JTAG)
  - In debug (JTAG open), keys in fuses and secrets in NVM are not available & CSME SRAM is zeroed
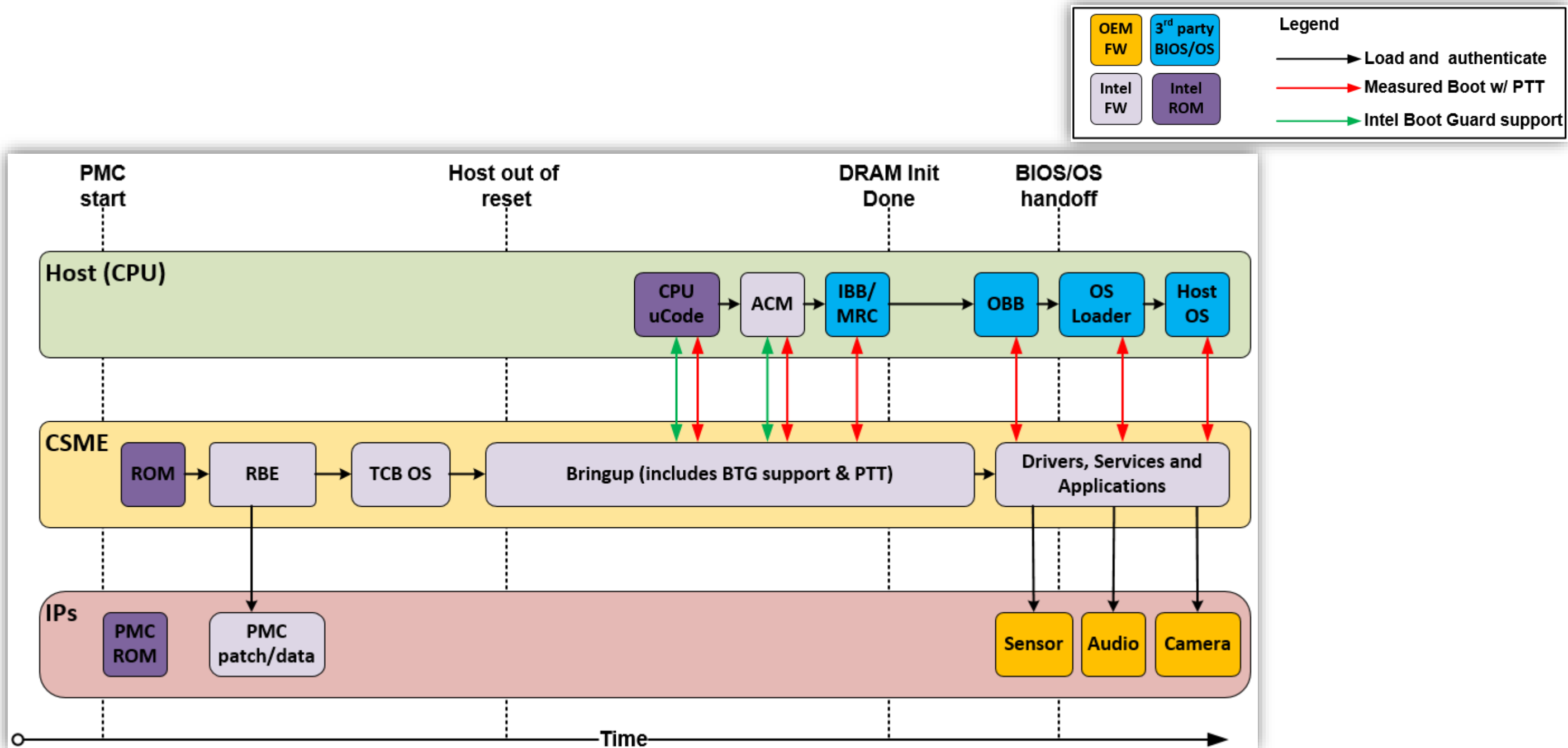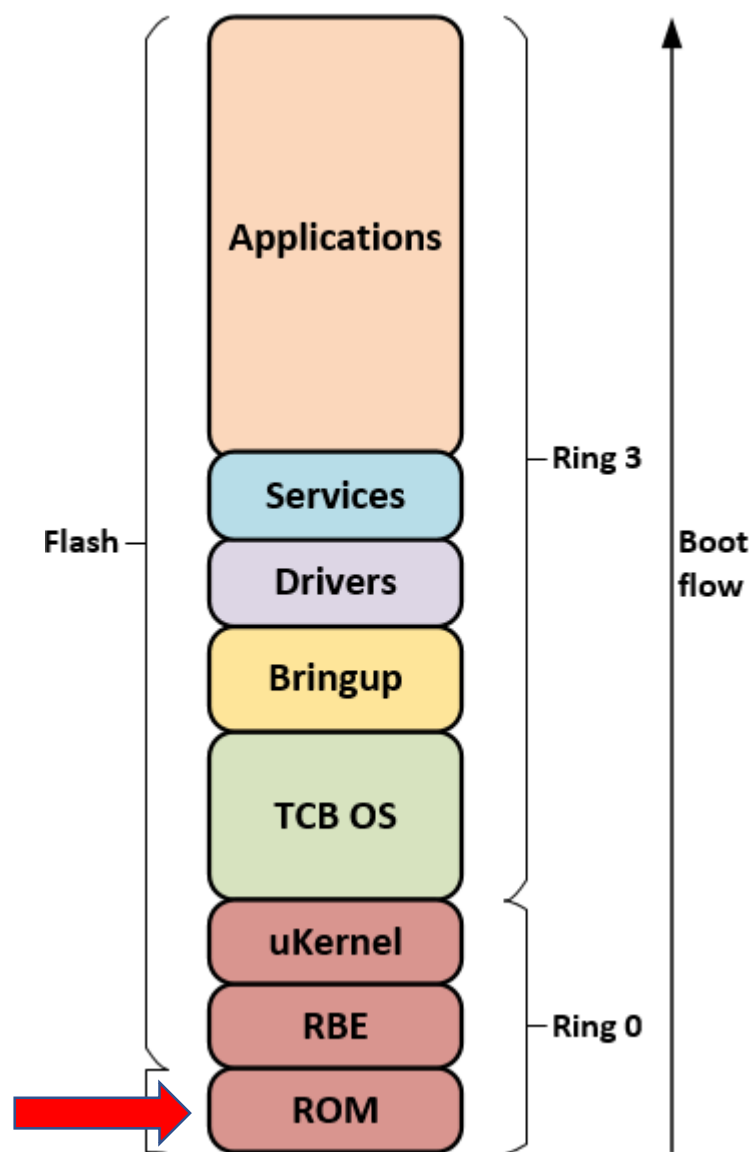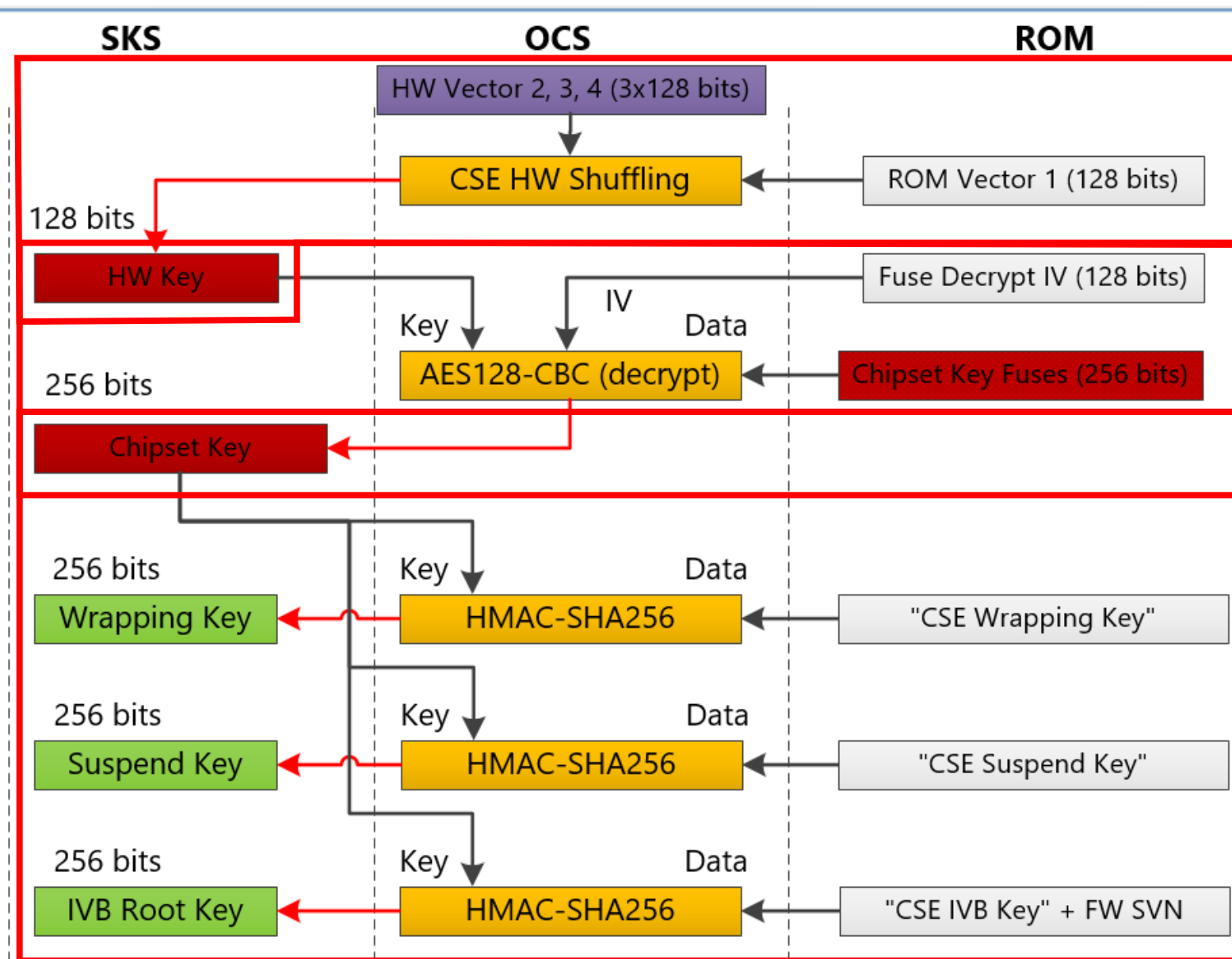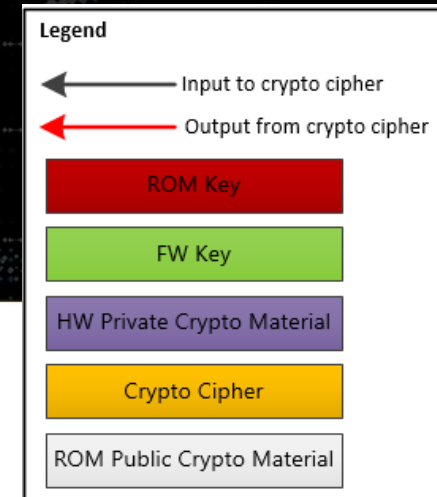
- **ROM is part of PCH HW with no patch mechanism after HW tape-in**
  - ROM bypass disabled by Intel manufacturing fuse on production stepping
- **Main responsibilities**
  - Moves CSME CPU to protected mode & enable paging and segmentation
  - Generates CSME FW keys using chipset key and RBE Security Version Number (TCB SVN)
  - Loads, authenticates and executes IDLM (debug module) / RBE
    - Hashes of public keys embedded in ROM
    - Intel manufacturing fuse indicates which public key is enabled (debug signing key is disabled on production)
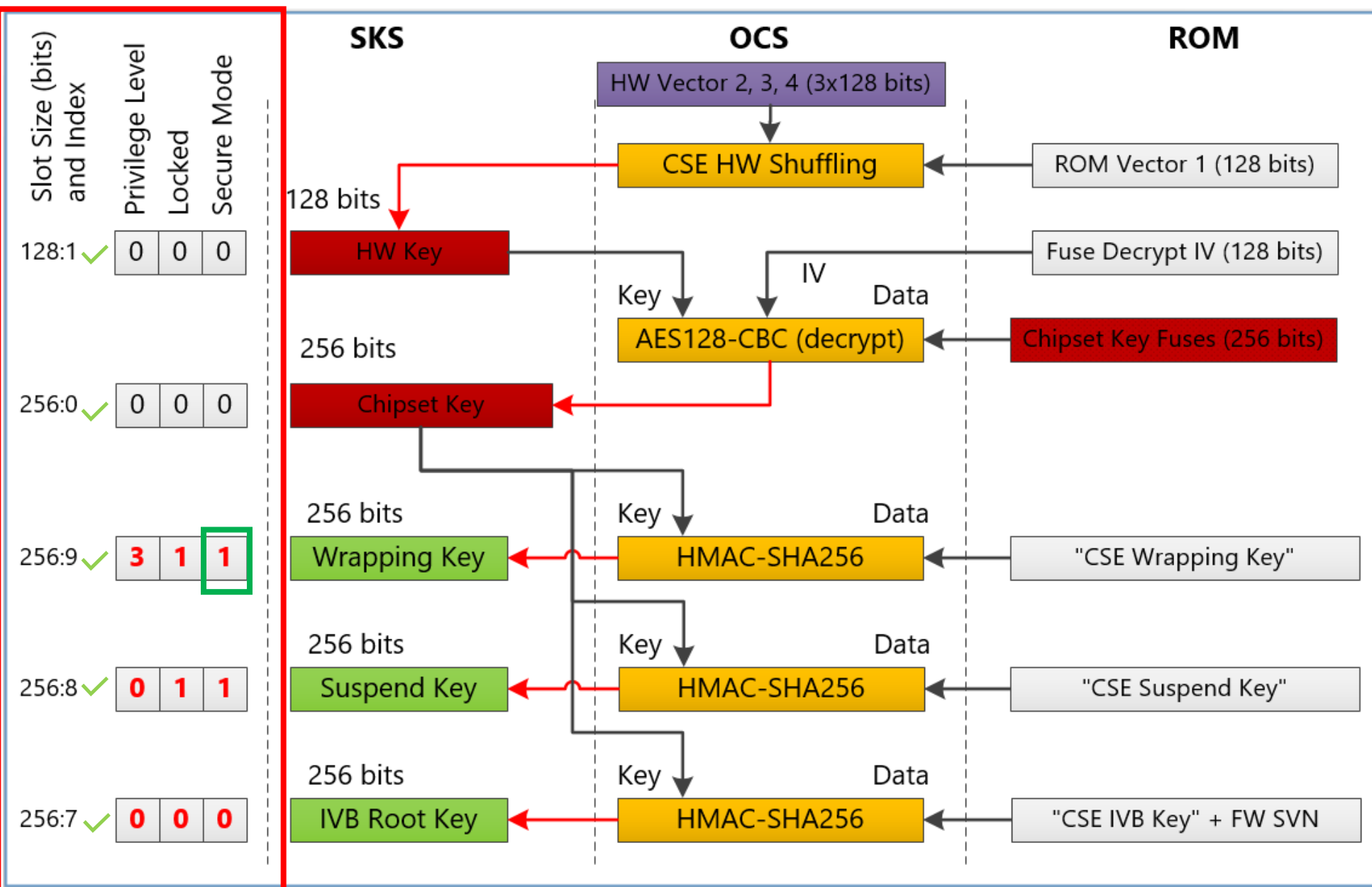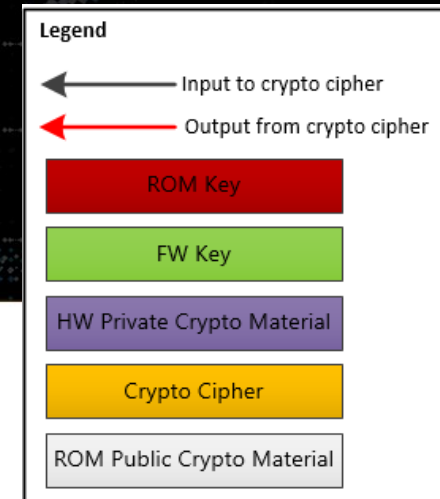
Key Derivation By ROM

Legend
- → Input to crypto cipher
- → Output from crypto cipher
- ROM Key
- FW Key
- HW Private Crypto Material
- Crypto Cipher
- ROM Public Crypto Material

SKS	OCS	ROM

- HW Vector 2, 3, 4 (3x128 bits)
- CSE HW Shuffling
- ROM Vector 1 (128 bits)
- 128 bits
- HW Key
- Fuse Decrypt IV (128 bits)
- Key — IV — Data
- 256 bits
- AES128-CBC (decrypt)
- Chipset Key Fuses (256 bits)
- Chipset Key
- 256 bits — Wrapping Key
- Key — Data
- HMAC-SHA256
- "CSE Wrapping Key"
- 256 bits — Suspend Key
- Key — Data
- HMAC-SHA256
- "CSE Suspend Key"
- 256 bits — IVB Root Key
- Key — Data
- HMAC-SHA256
- "CSE IVB Key" + FW SVN

1. HW key generation

2. Decryption of Chipset Key

3. Derivation of CSME FW Key
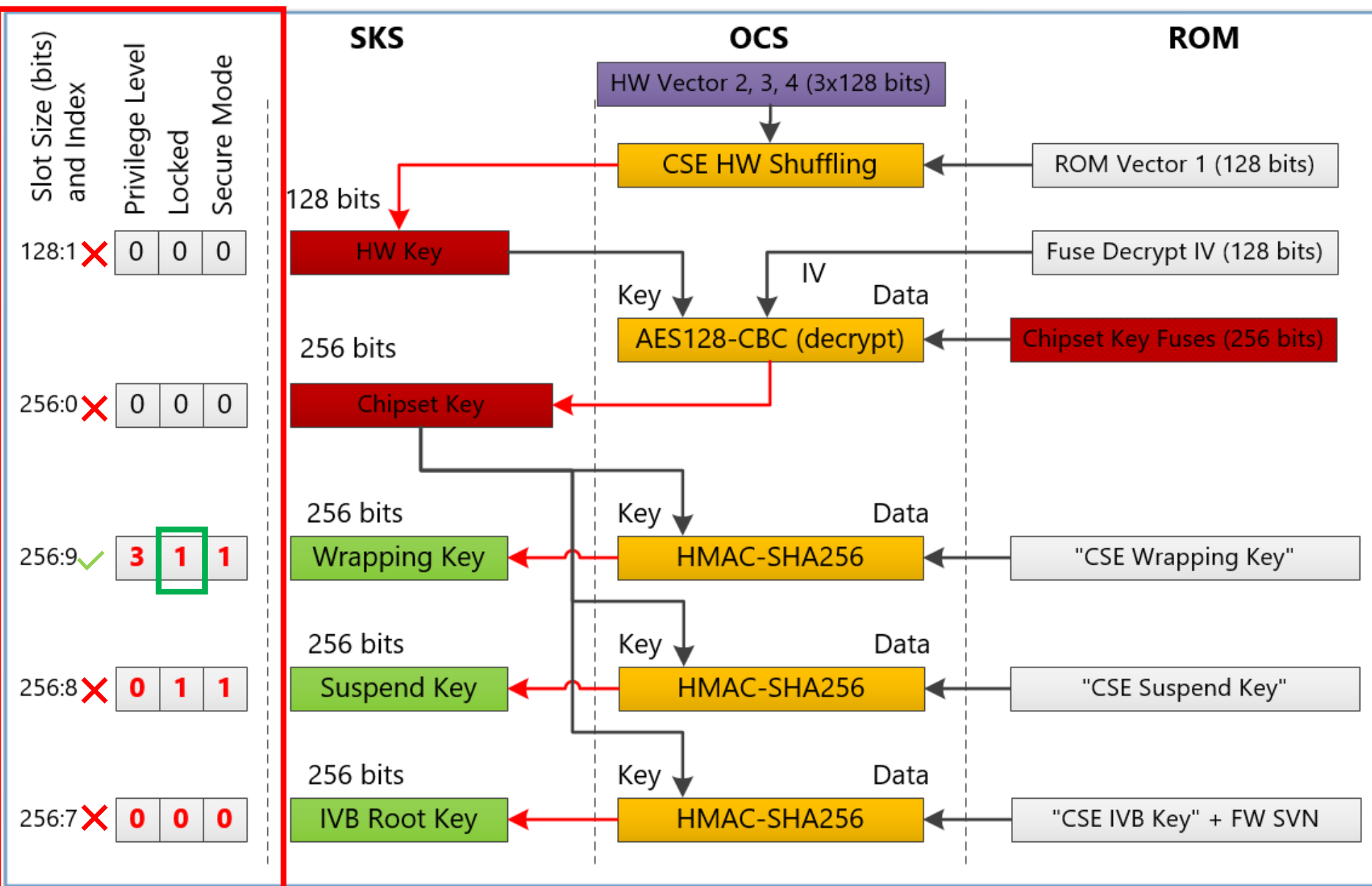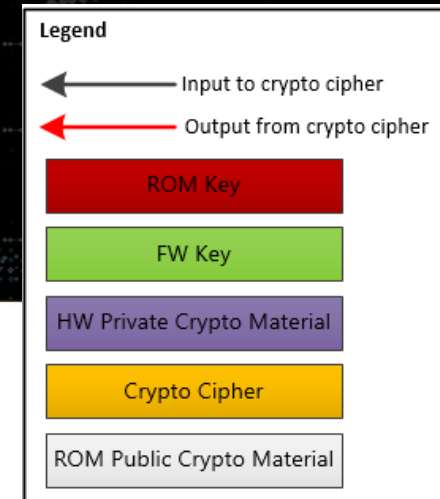
HW Secure Key Storage

HW SKS:

Protect CSME root keys during runtime. FW can only use keys

Every SKS slot has set of attributes

- Secure Mode
  - Result of AES-CBC decrypt and HMAC using the key in this slot can be stored in SKS only

- Privilege Level: used for HW access control on SKS slot
  - The key in this slot is accessible if SKS slot privilege level is >= SKS privilege level

SKS Privilege Level 0

HW Secure Key Storage

Legend
- Input to crypto cipher
- Output from crypto cipher
- ROM Key
- FW Key
- HW Private Crypto Material
- Crypto Cipher
- ROM Public Crypto Material

SKS | OCS | ROM

HW Vector 2, 3, 4 (3x128 bits)

CSE HW Shuffling ← ROM Vector 1 (128 bits)

128 bits — HW Key

Fuse Decrypt IV (128 bits)

Key / IV / Data

AES128-CBC (decrypt) ← Chipset Key Fuses (256 bits)

256 bits — Chipset Key

256 bits — Wrapping Key ← HMAC-SHA256 ← "CSE Wrapping Key"

256 bits — Suspend Key ← HMAC-SHA256 ← "CSE Suspend Key"

256 bits — IVB Root Key ← HMAC-SHA256 ← "CSE IVB Key" + FW SVN

Slot Size (bits) and Index | Privilege Level | Locked | Secure Mode

| Slot	Privilege Level	Locked	Secure Mode
128:1 ✗	0	0	0
256:0 ✗	0	0	0
256:9 ✓	3	1	1
256:8 ✗	0	1	1
256:7 ✗	0	0	0

HW SKS:

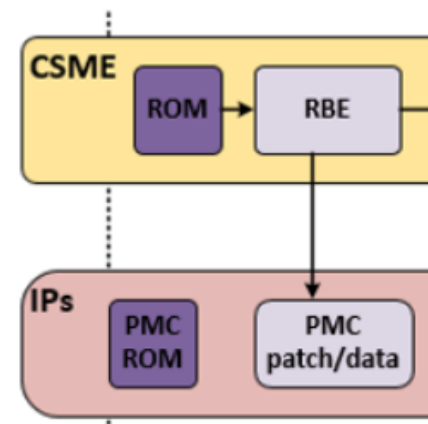Protect CSME root keys during runtime. FW can only use keys

Every SKS slot has set of attributes

- Secure Mode
  - Result of AES-CBC decrypt and HMAC using the key in this slot can be stored in SKS only

- Privilege Level: used for HW access control on SKS slot
  - The key in this slot is accessible if SKS slot privilege level is >= SKS privilege level

- Locked: key in this slot can be invalidated or replaced after CSME HW reset only

SKS Privilege Level **3**

#BHUSA @BLACK HAT EVENTS
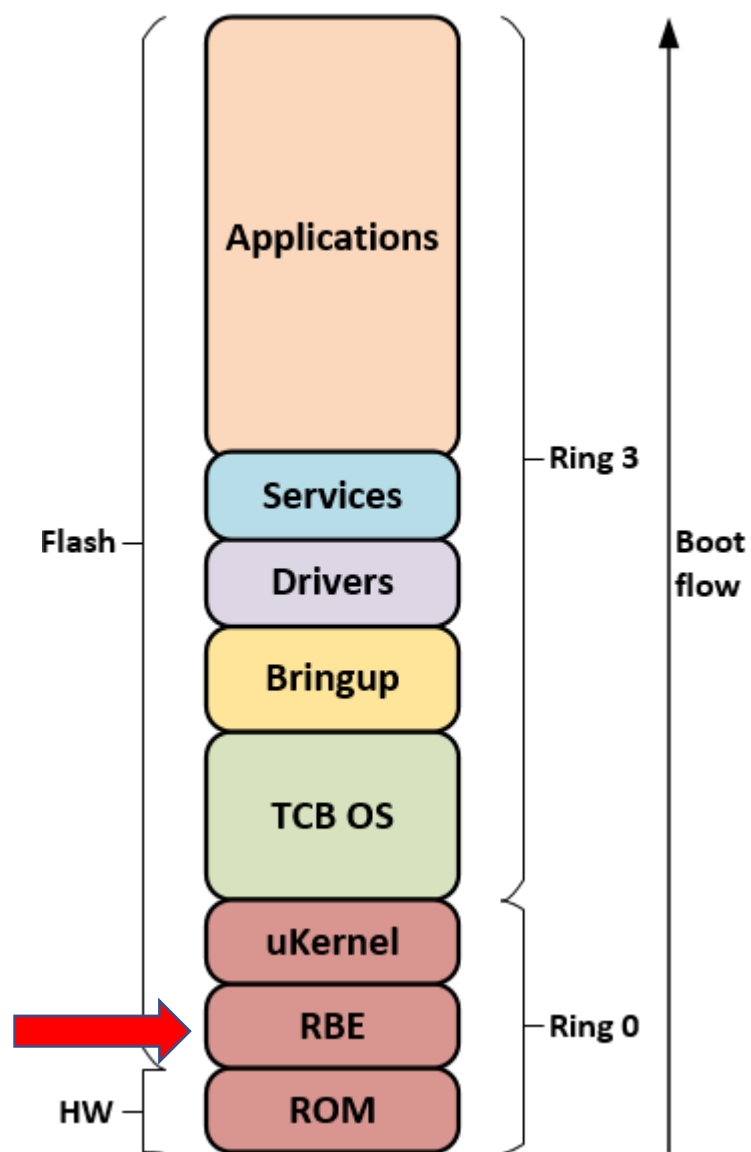
- Extends ROM functionality in FW (can be updated on field)
- Bootloader of CSME OS
- Main responsibilities
  - Performs HW based anti-rollback check on CSME FW
  - Performs early chassis job – PMC patch
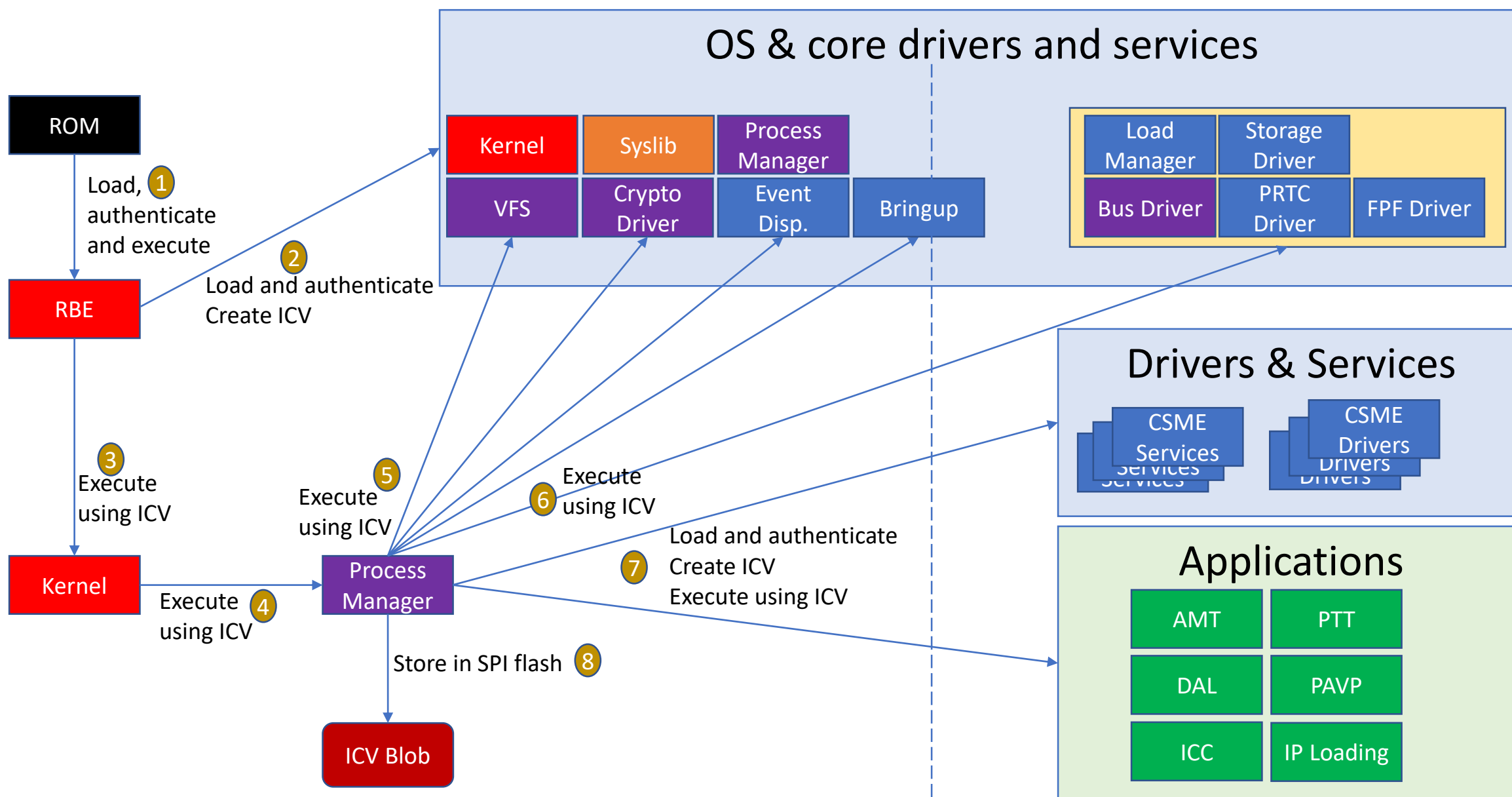 - Loads, authenticates and executes CSME OS

OS & core drivers and services

Kernel	Syslib	Process Manager
VFS	Crypto Driver	Event Disp.	Bringup

Load Manager	Storage Driver	
Bus Driver	PRTC Driver	FPF Driver

Legend
ICV: Integrity Check Value

FW Ring0	FW TCB Ring3
HW Ring0	FW OS Ring3
FW App Ring3	FW lib. Ring3

ROM

Load, ① authenticate and execute

RBE → ② ICV Blob
Load and authenticate

③ Execute using ICV

Kernel → ④ Execute using ICV → Process Manager

⑤ Execute using ICV

⑥ Execute using ICV

⑦ Execute using ICV

Drivers & Services

CSME Services
Services

CSME Drivers
Drivers

Applications

AMT	PTT
DAL	PAVP
ICC	IP Loading

ICVs are used to speed-up CSME boot

time

DRAM Init Done

Code pages replacement using ICV done by kernel from SPI flash

Code and data pages replacement using ICV done by kernel from DRAM
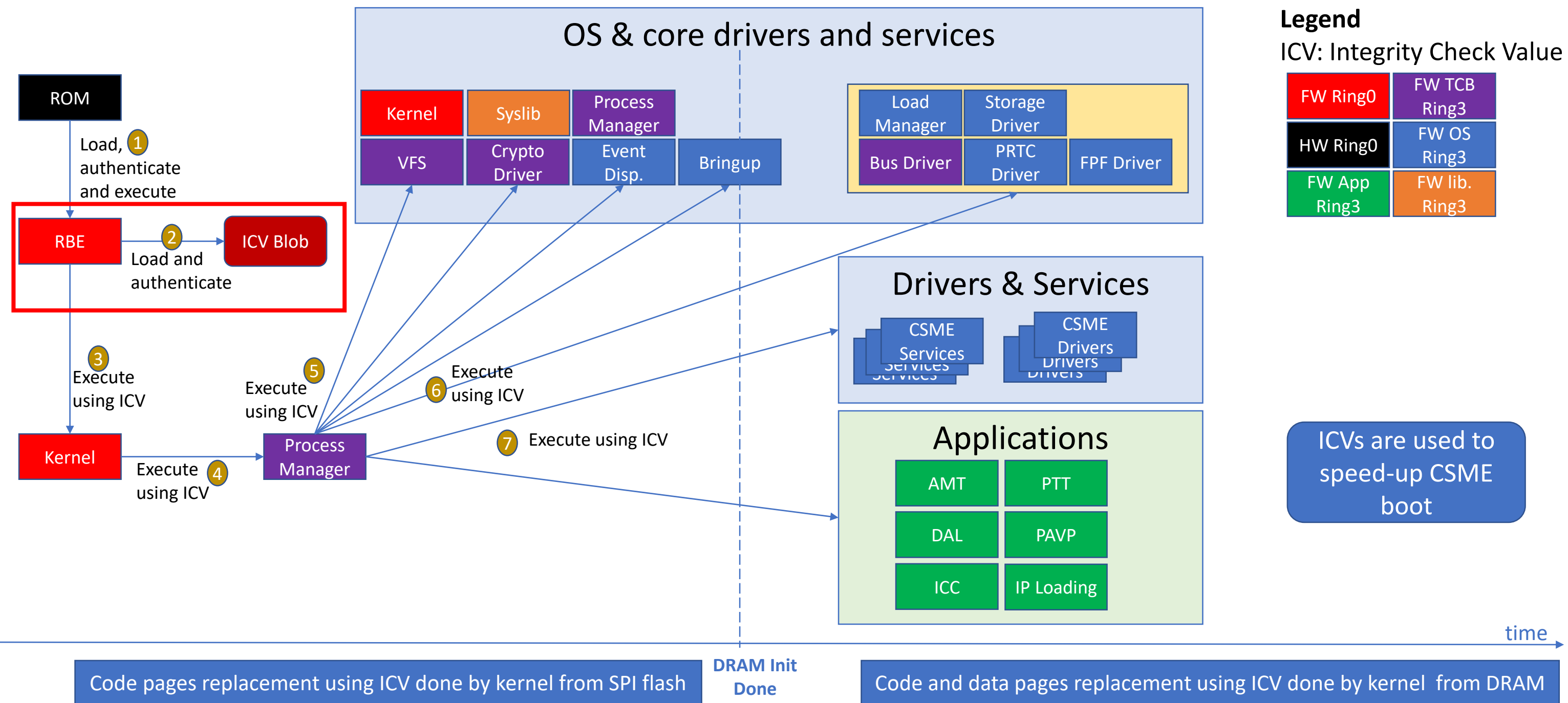
- Architecture & Boot flow
- **OS Security Principles & Internals**
- Hardening & Mitigations
- Pre & Post Manufacturing
- Update & Recoverability
- Wrap-up

- ## Micro-Kernel OS based on Minix OS architecture
  - The micro-kernel is the only runtime component running at ring0. Application, Drivers and Services run at ring3
  - The micro-kernel implements the bare minimum required to implement an OS

- ## Minimal Trusted Compute Base (TCB)
  - Protects access to keys and HW (CSME assets)
  - Responsible for CSME FW code integrity at boot & runtime
  - Responsible for protection CSME modules from each other and their data in SPI flash
  - Enforces CSME modules' minimum privileges
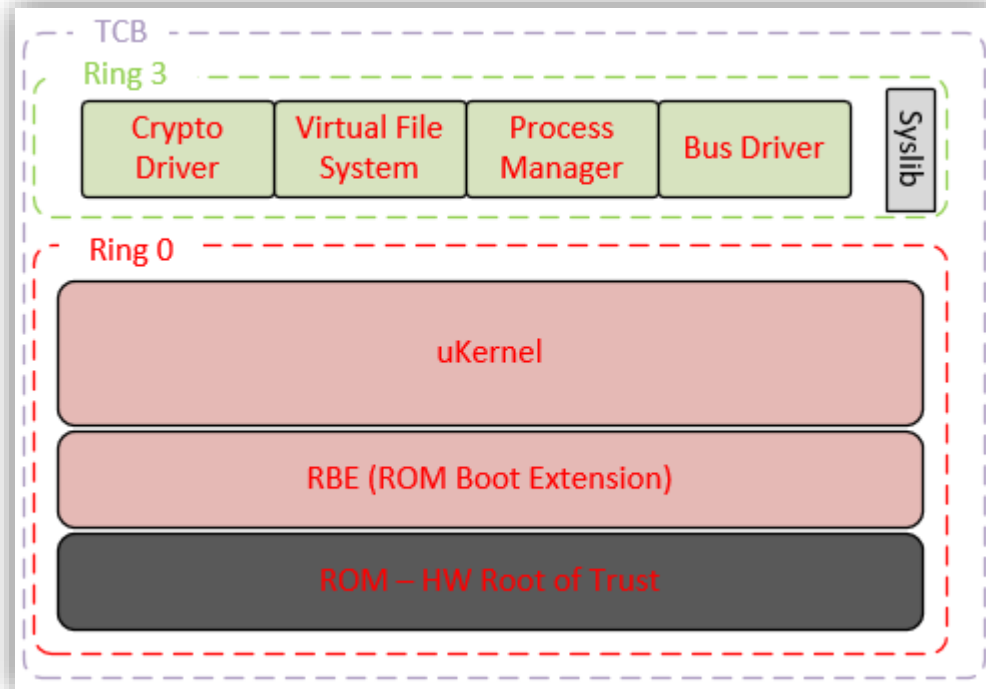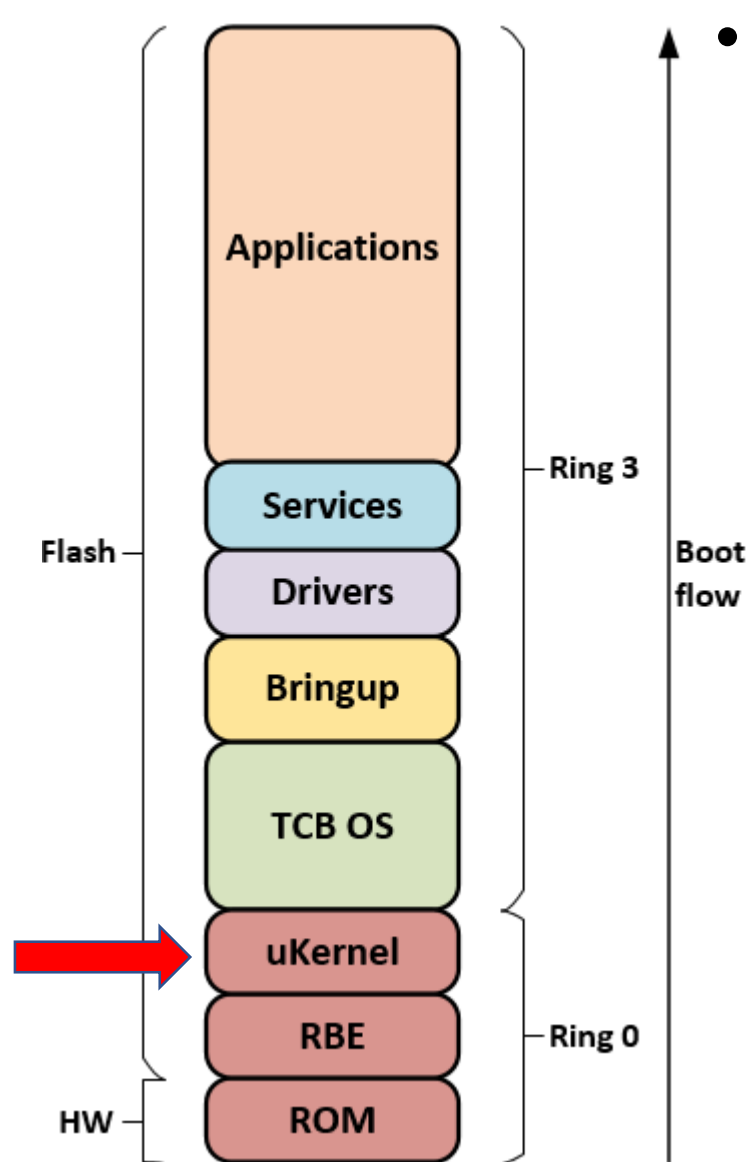
- ## Main responsibilities
  - ### Driver of the CPU
    - Enforces code execution is from SRAM only
    - Enforces process isolation using CPU rings and x86 segments
    - Setups page attributes (RW and User bit). Enforcement done by MMU
    - Controls HW access via MMIO by ring3
  - ### Driver of the IOMMU
    - Controls DMA access to SRAM
  - ### Support standard kernel service
    - Inter-Process Communication (IPC)
    - Processes & threads management
    - Interrupts and exceptions handling
  - ### Handle page replacement between SRAM and DRAM/SPI flash to save SRAM utilization
    - Evicted pages to DRAM are encrypted and integrity protected
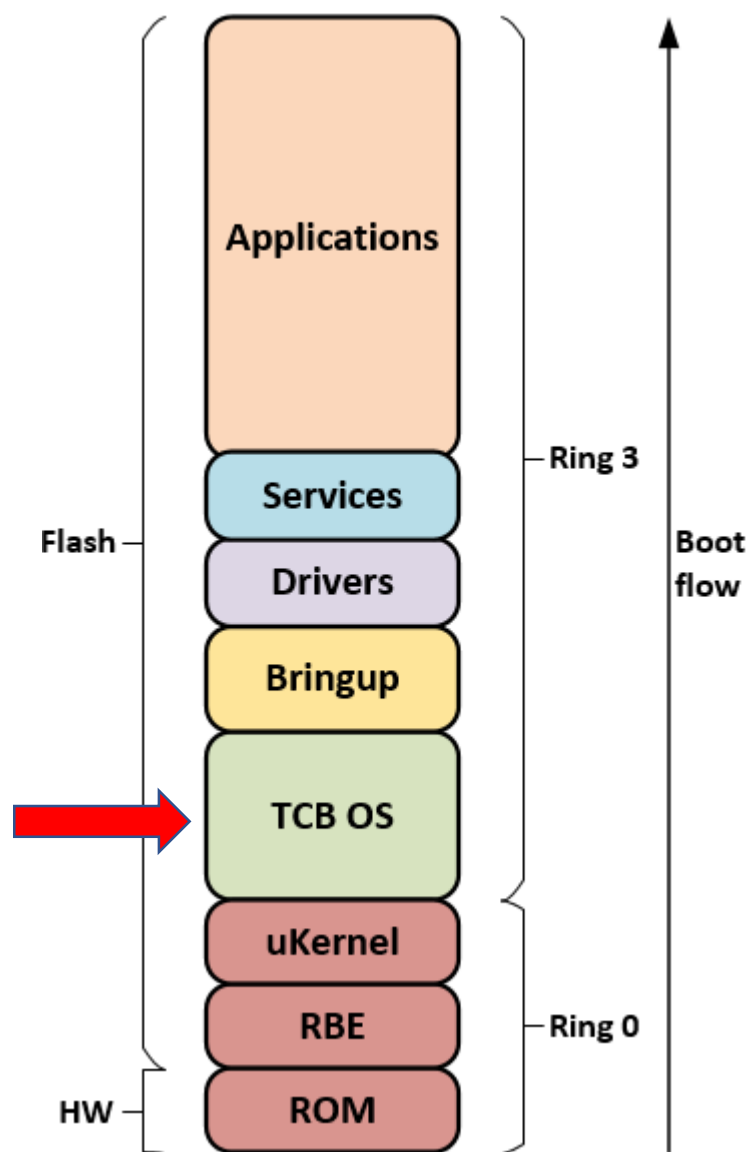
| Ring3 TCB Component	Main Security Role	Create CSME Process	Control access to CSME keys	Control access to CSME OS services	Control access to Hardware
Process Manager	1. Code authentication 2. Process creation & termination	**Yes**	No	No	No
Crypto Driver	1. Crypto & DMA service 2. FW key management	No	**Yes**	No	No
Virtual File System (VFS)	1. Secure storage service with data migration support 2. Enforce permission check on data files and special files exposed by drivers & services	No	No	**Yes**	No
Bus Driver	1. Allow CSME drivers to configure their own device configuration space 2. Enforce access control	No	No	No	**Minimal**

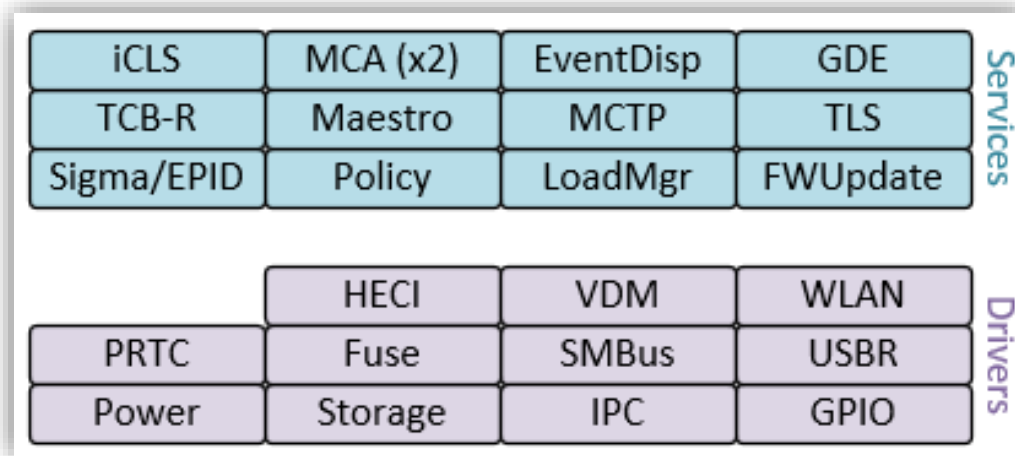- Support early platform boot and configuration
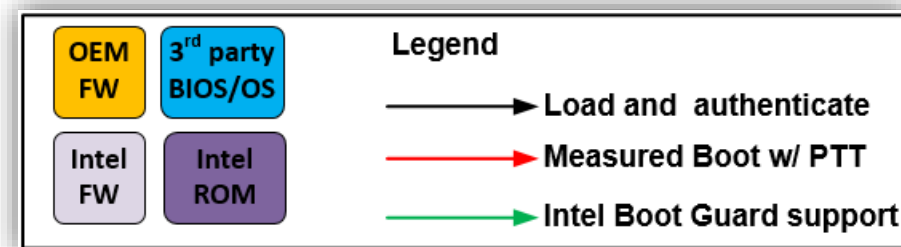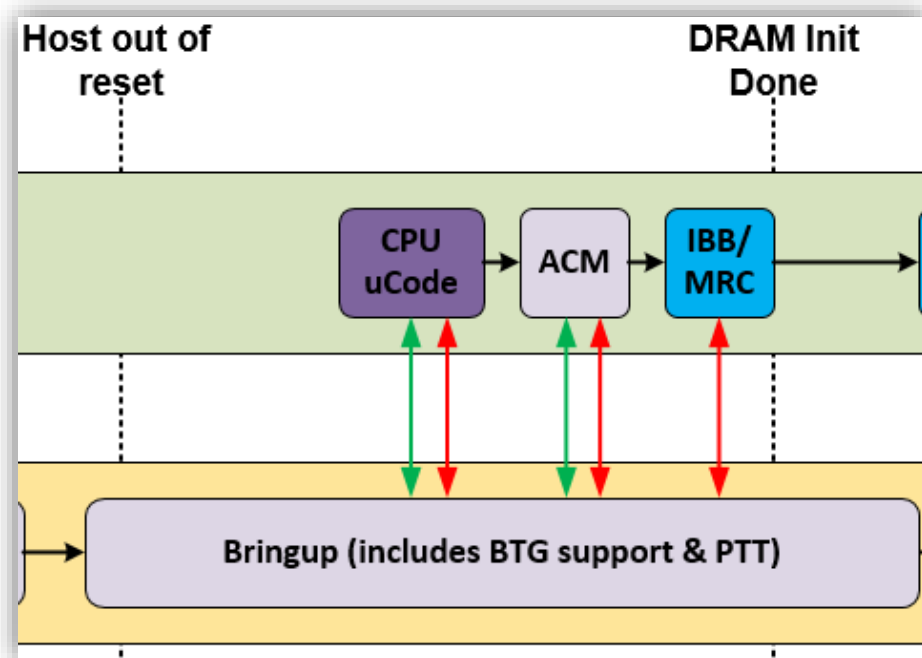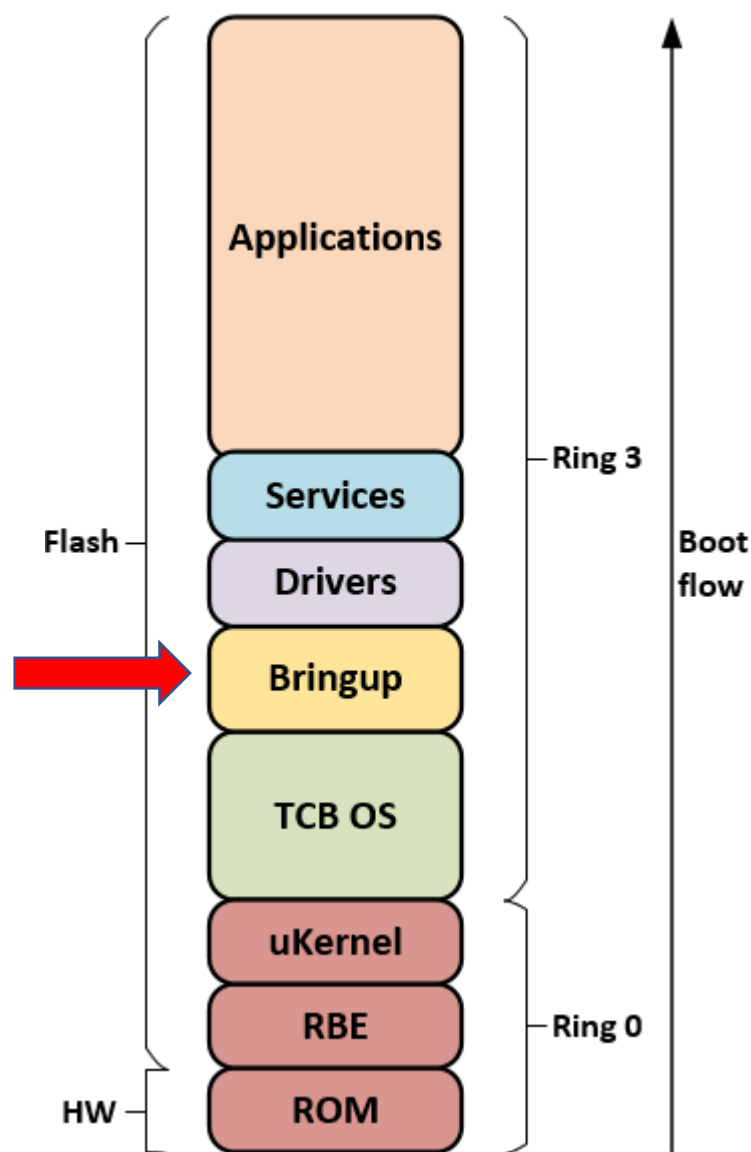
- Reduced its privileges starting CSME 12
  - Can't create CSME processes
  - No access to root keys and attestation keys
  - No access to crypto accelerator and DFX HW
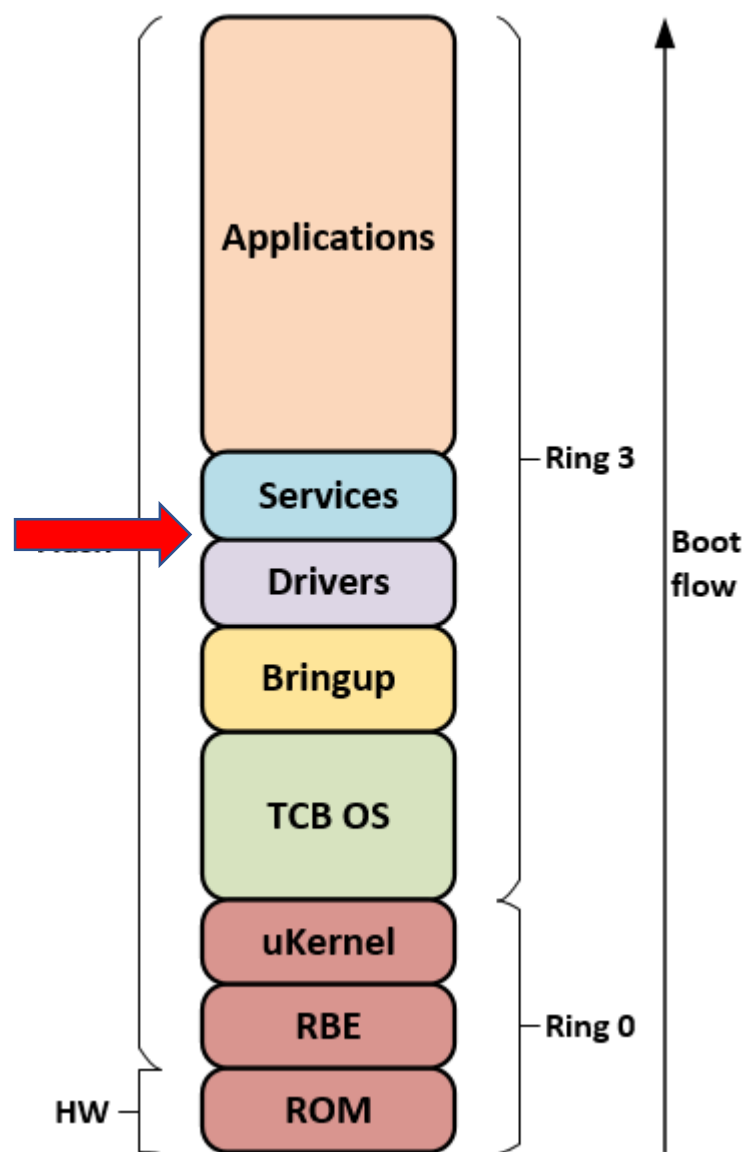
- Driver and Services are running at ring3
- Drivers have access only to HW they need to manage via Memory Mapped IO (MMIO)

- Access to drivers and services are jointly controlled by VFS & uKernel

Services:
iCLS: Intel Capability License Server protocol
Event Dispatcher: Publish system wide events
MCA: Support manufacturing flows
GDE: Display overlay used by AMT for user consent
TCB-R: Support for EPID re-key over iCLS
Maestro: Coordinate power state transitions
MCTP: Implementation of DMTF MCTP protocol
TLS: Standalone TLS stack (used by DAL)
Sigma/EPID: SIGn and Message Authentication protocol using Intel EPID
Policy: CSME features enable/disable
LoadMgr: Authentication service and boot order
FWUpdate: In-band FW Update

Drivers:
PRTC: Protected Real Time Clock
VDM: CPU and GFX Communications
WLAN: AMT WLAN OOB driver over Clink
IPC: IP to IP communication (PMC, ISH etc.)
Fuse: FPF driver supporting read/write
SMBus: SMBus/I2C interface
USBR: AMT redirection (keyboard, mouse, storage)
HECI: Host interface driver
Storage: SPI driver and low level filesystem
Power: Power management; PMC communication
GPIO: GPIO configuration and usage

- CSME applications are running at ring3
- CSME TCB ensure CSME applications are isolated from each others including their data kept in NVM
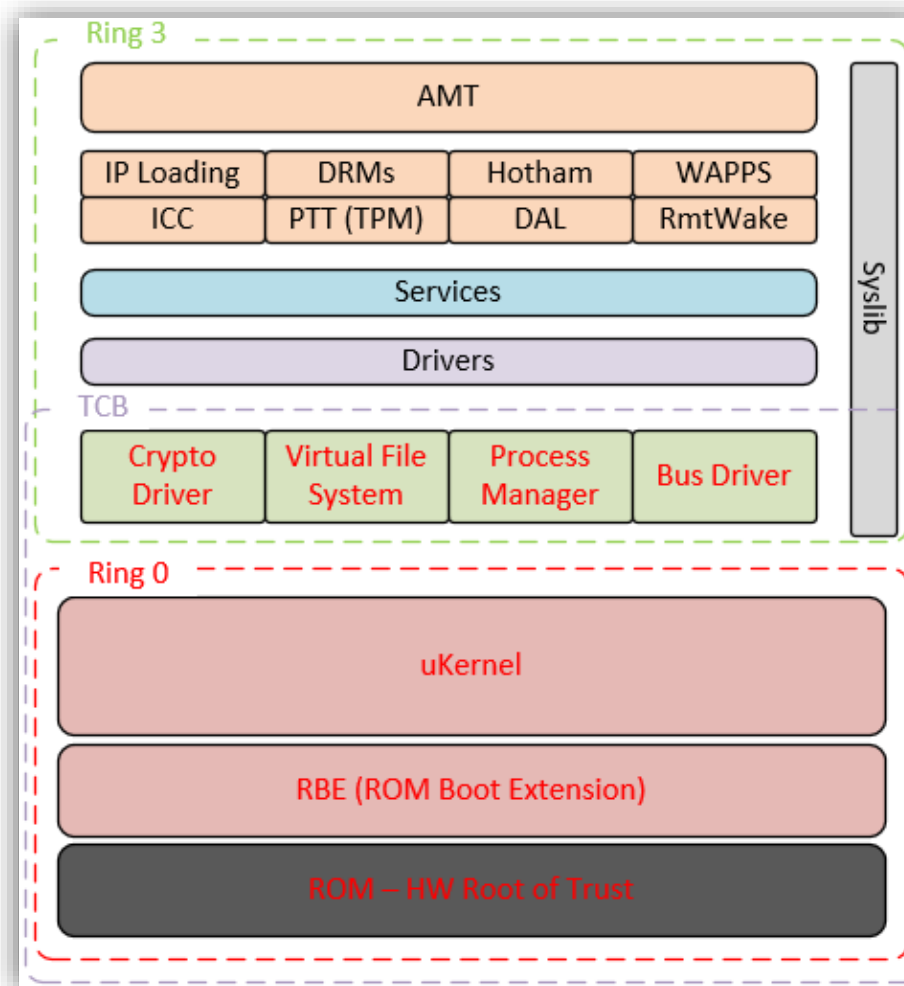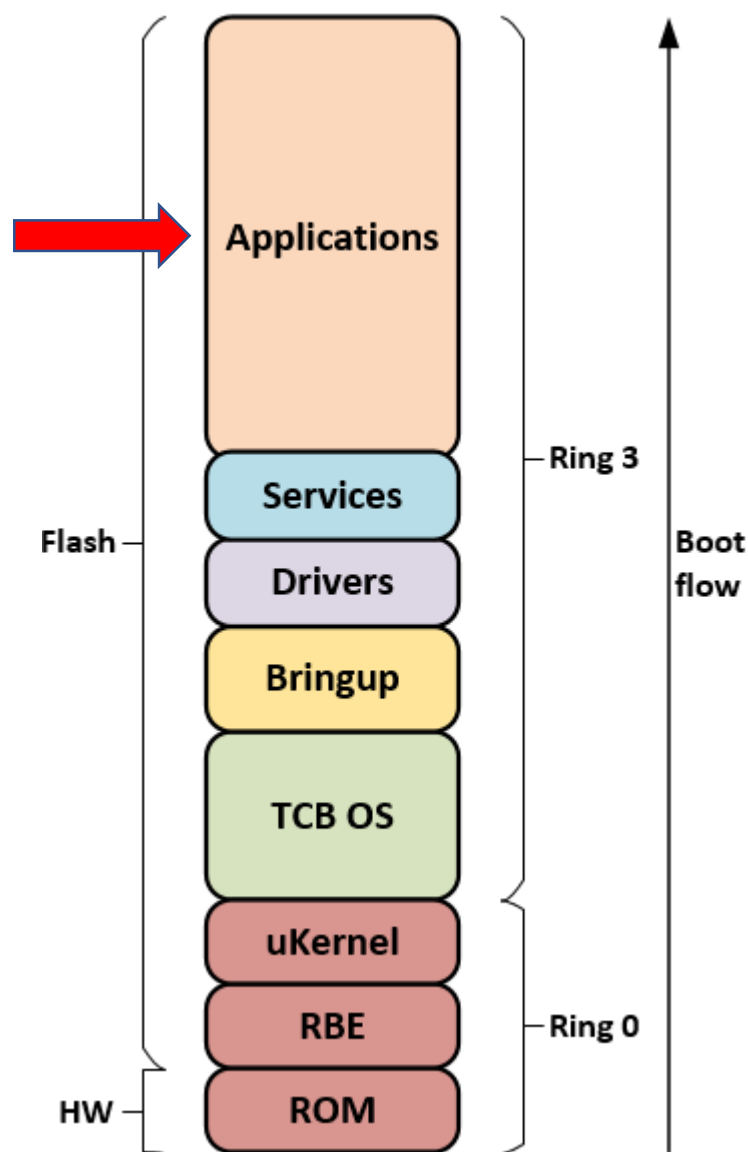
Applications:
AMT: Manageability including network stack
IP loading: ISH, Audio, Camera
PAVP: PlayReady, Widevine, HDCP
Hotham: Debug mailbox with SW
WAPPS: AMT 3rd party storage
ICC: Integrated Clock Configuration (overclocking)
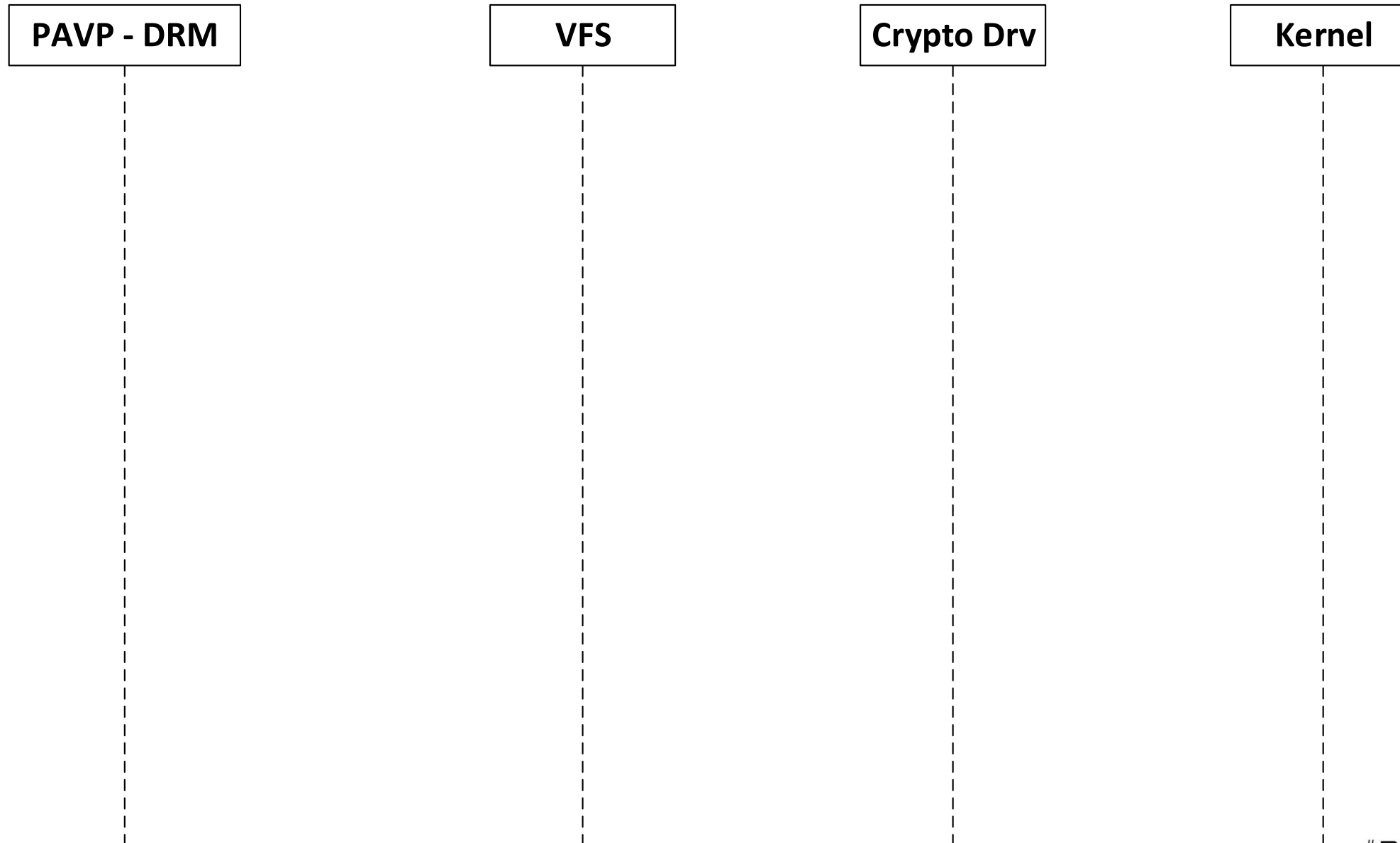**PTT**: TPM 2.0 implementation
DAL: Dynamic Intel signed applications loading
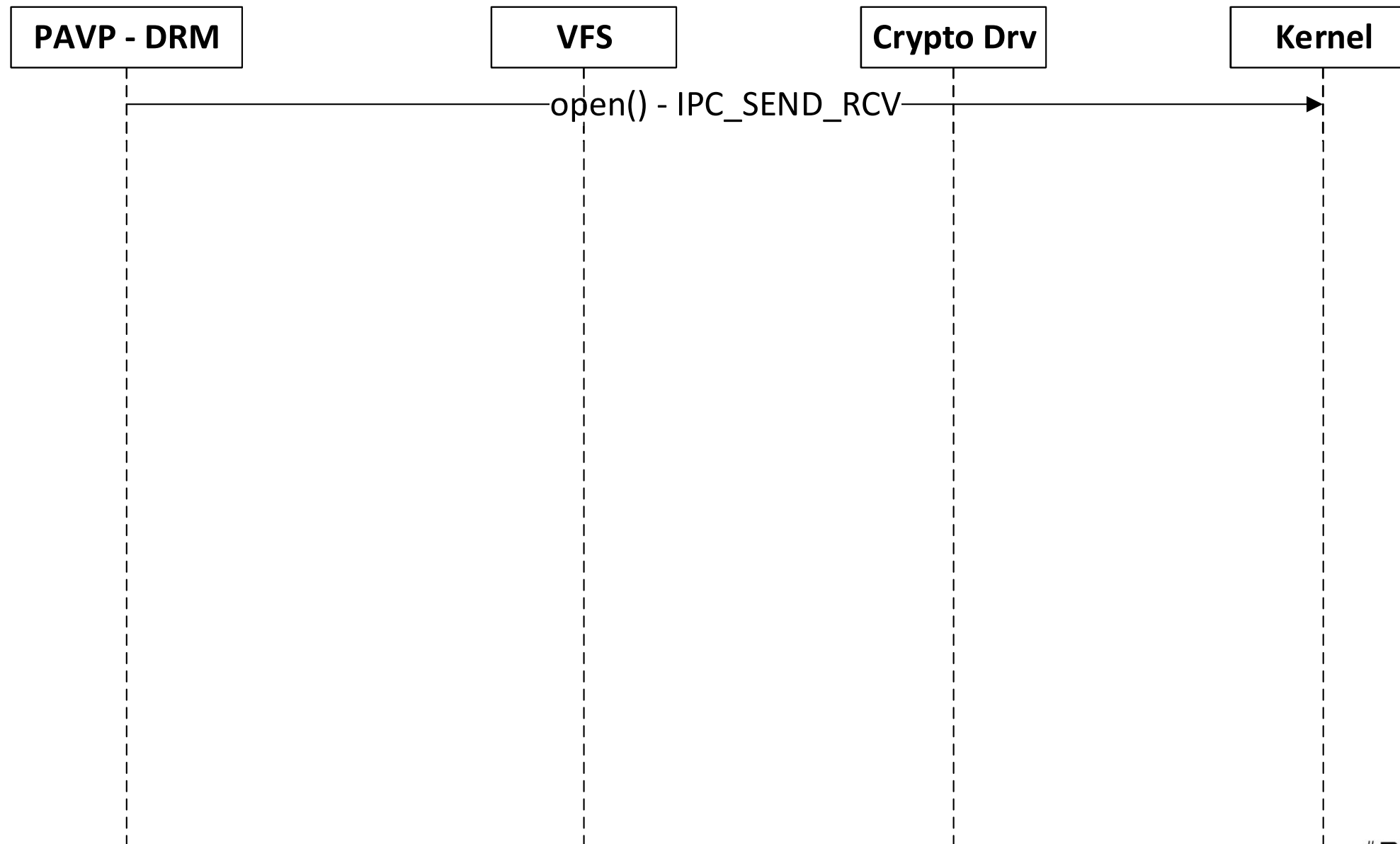**RmtWake**: Support for concurrent Wake On LAN

- Architecture & Boot flow
- OS Security Principles & Internals
- **Hardening & Mitigations**
- Pre & Post Manufacturing
- Update & Recoverability
- Wrap-up

- Kernel (Ring0)
  - Kernel system call filtering
  - Applied stack protector for kernel
  - Data execution prevention
  - Activated Supervisor Mode Access Prevention (SMEP)
  - Use CR0.Write.Protect
    - Prevent corruption of read only pages by kernel
  - ACL on Ring3 inter-process communication

- Example of authorized IPC (Abstracted)

| PAVP - DRM | | VFS | | Crypto Drv | | Kernel |

- Example of authorized IPC (Abstracted)

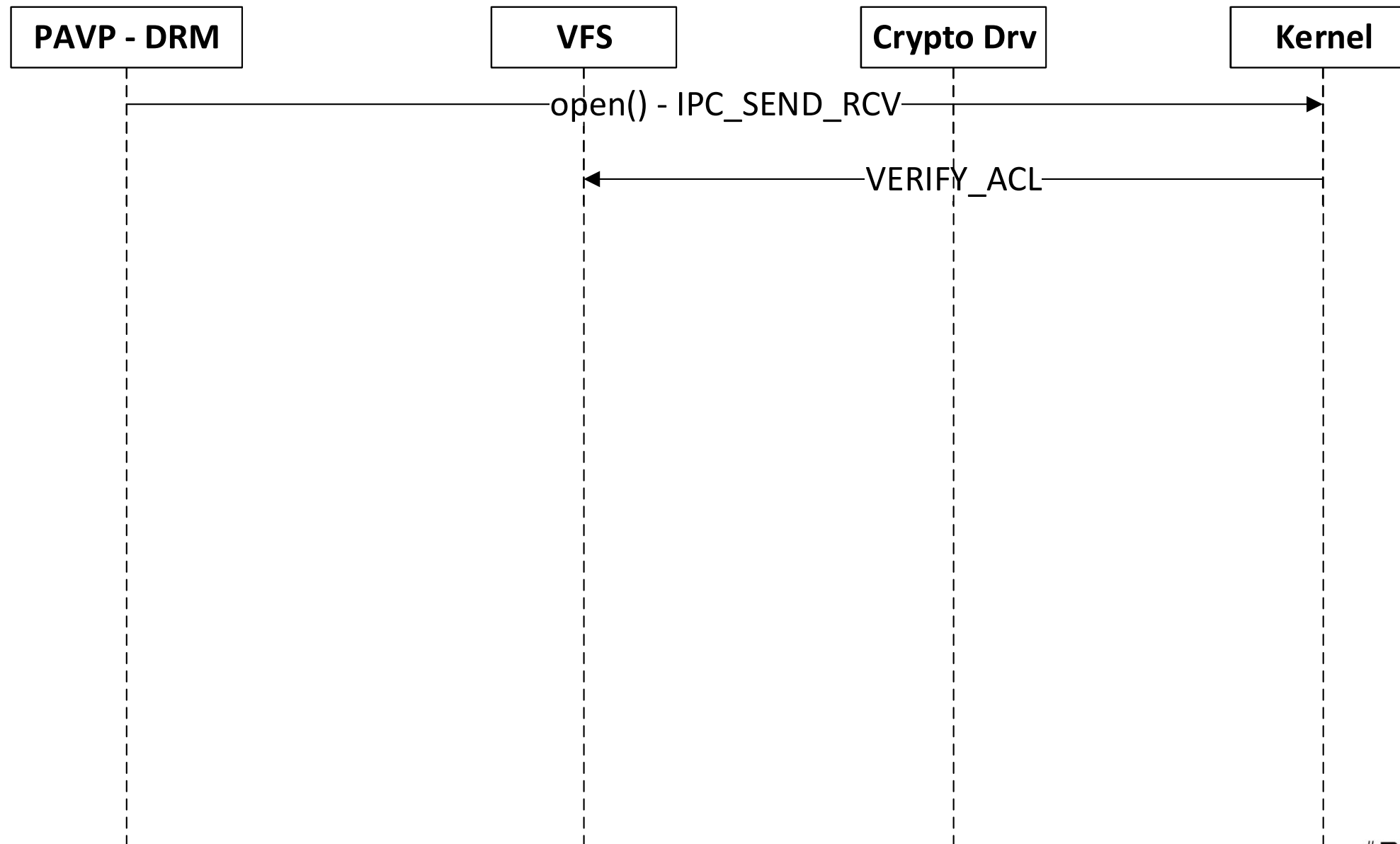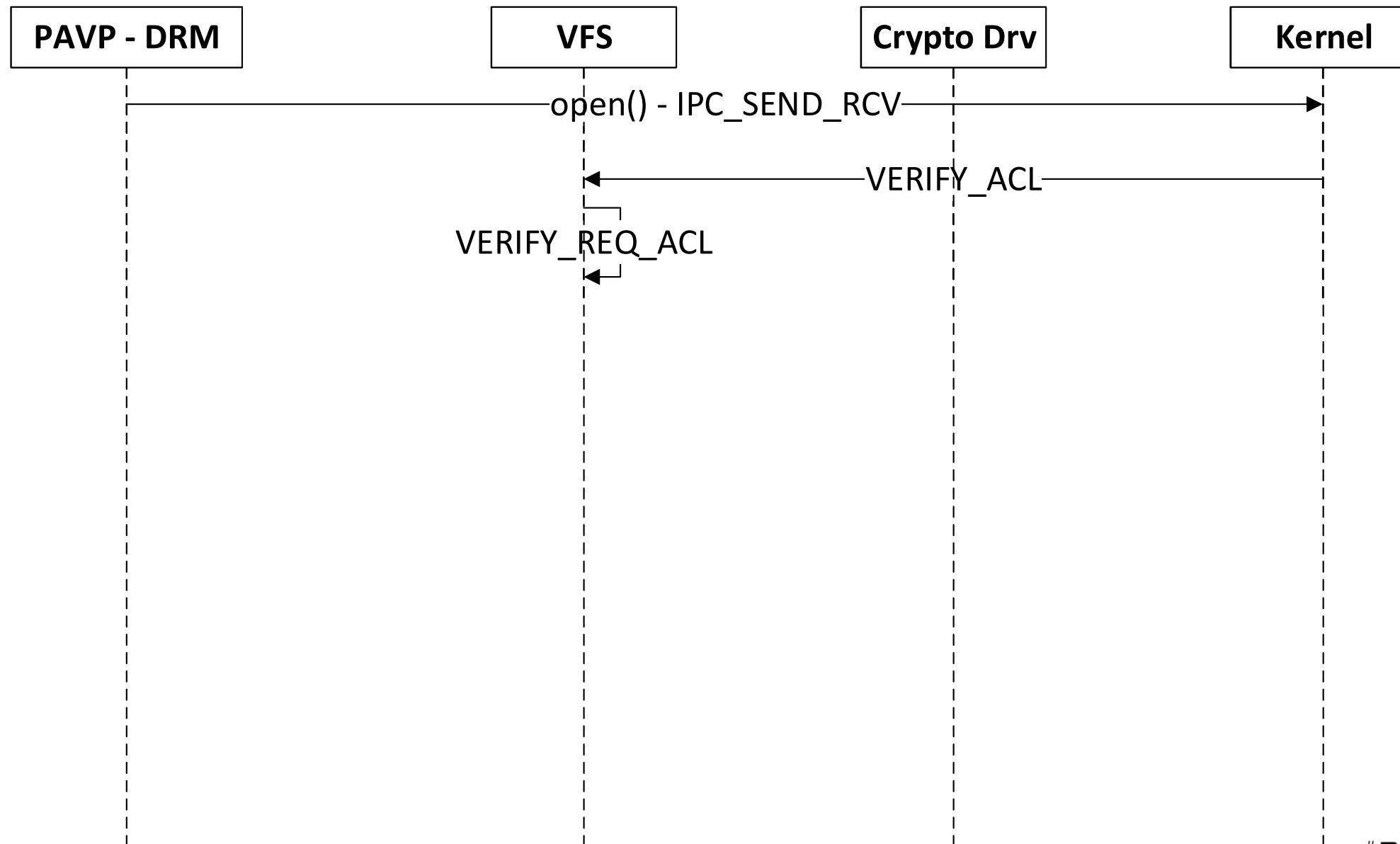| PAVP - DRM	VFS	Crypto Drv	Kernel

open() - IPC_SEND_RCV

- Example of authorized IPC (Abstracted)

- Example of authorized IPC (Abstracted)

- Example of authorized IPC (Abstracted)

- Example of authorized IPC (Abstracted)

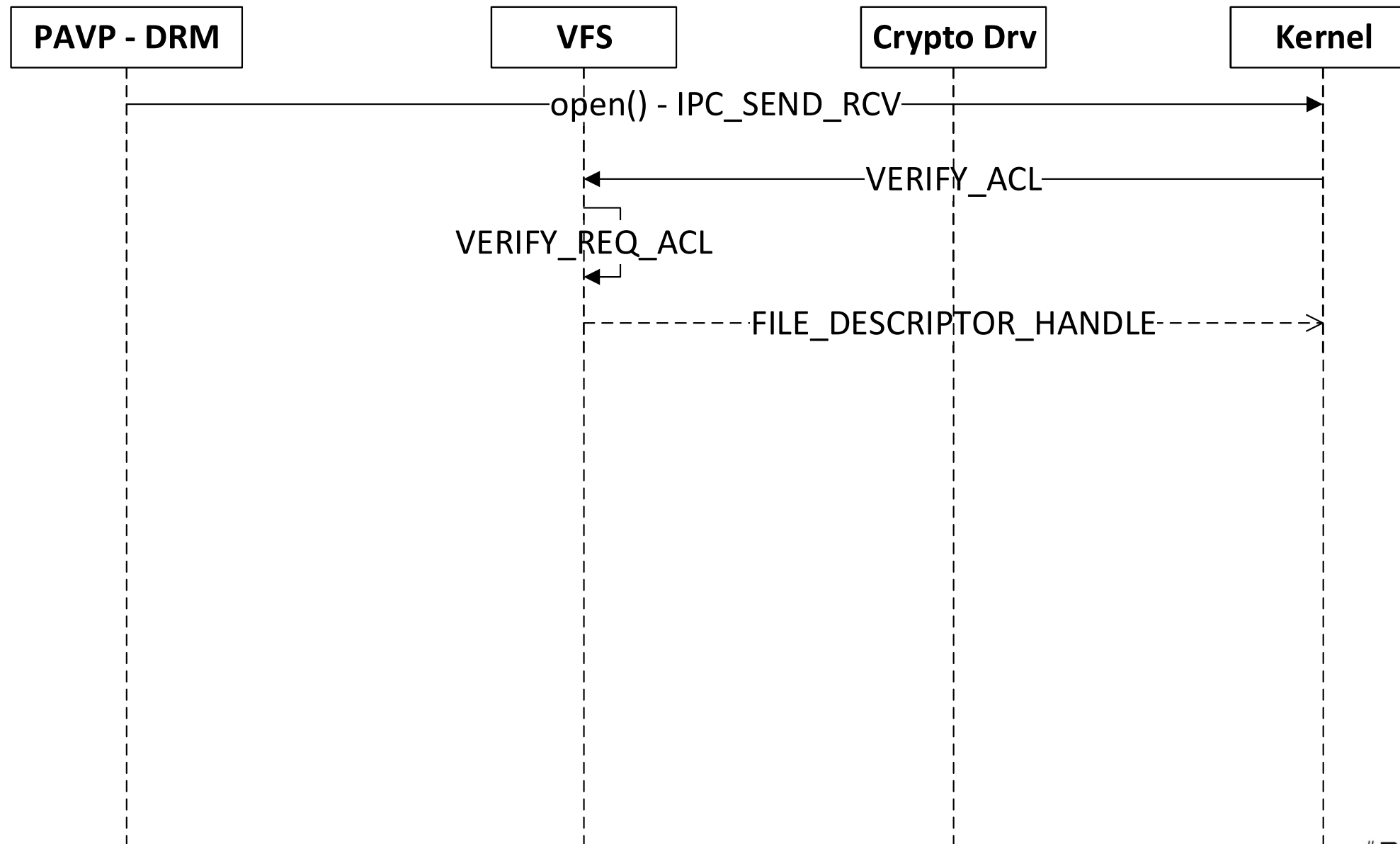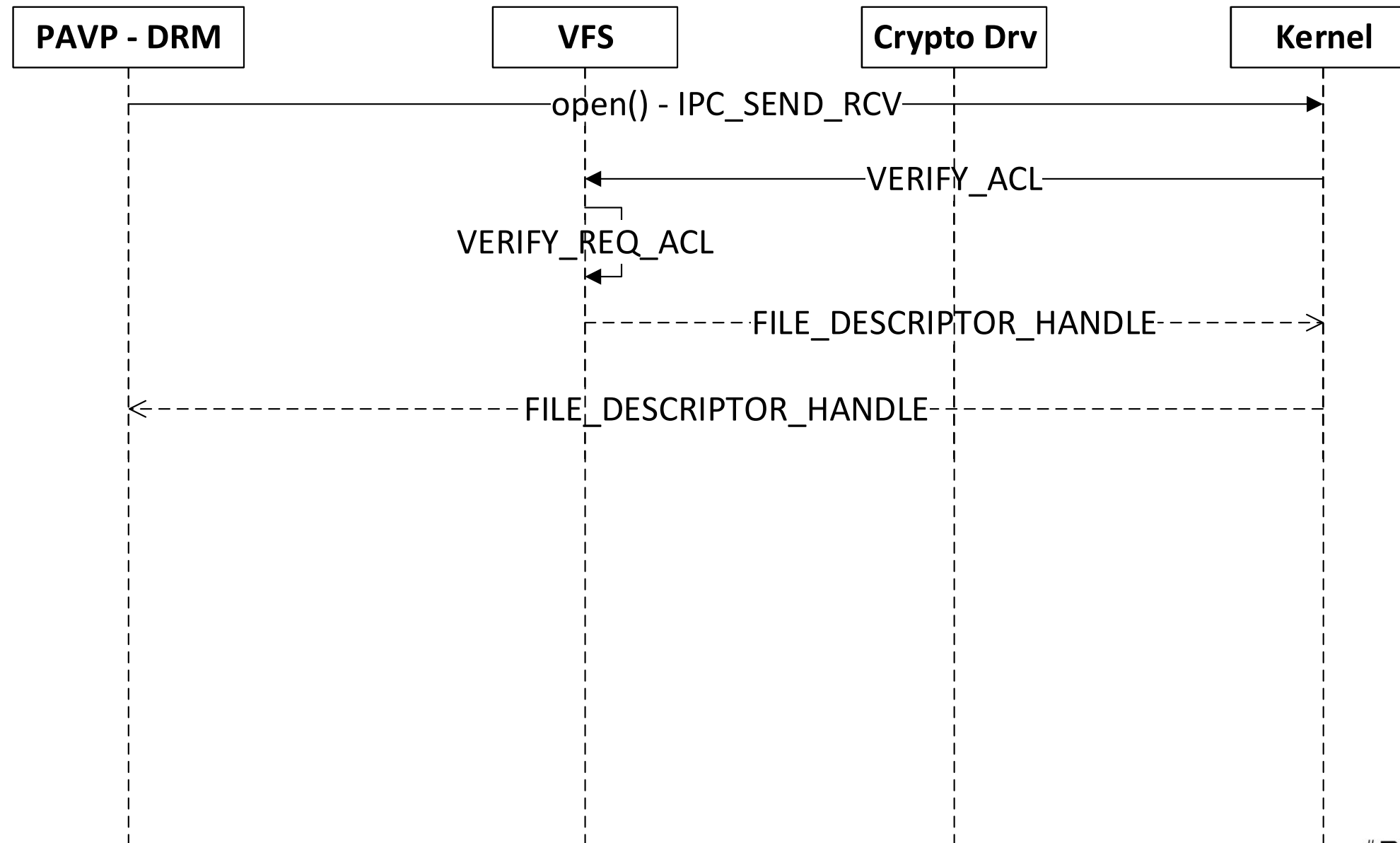- Example of authorized IPC (Abstracted)

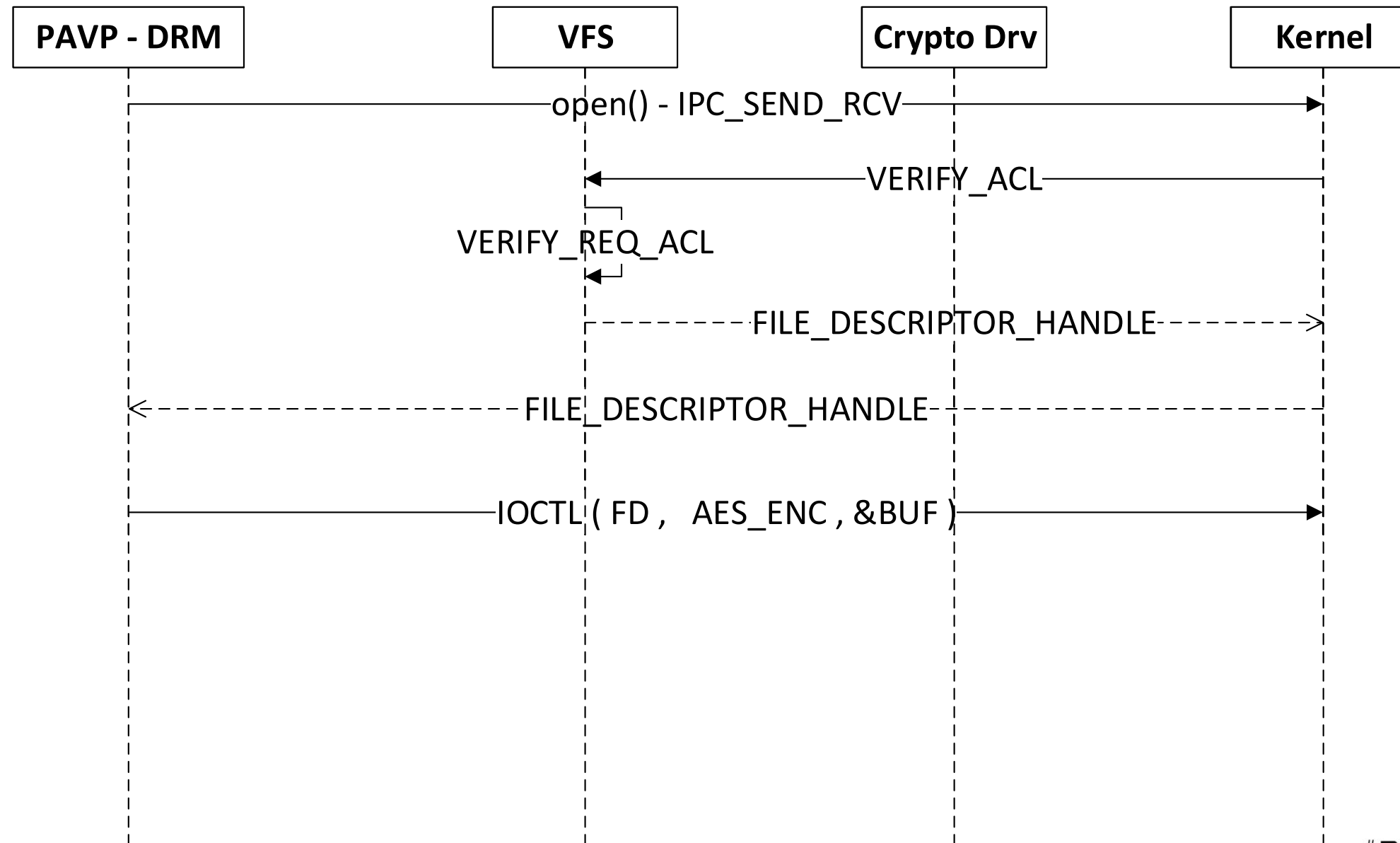- Example of authorized IPC (Abstracted)
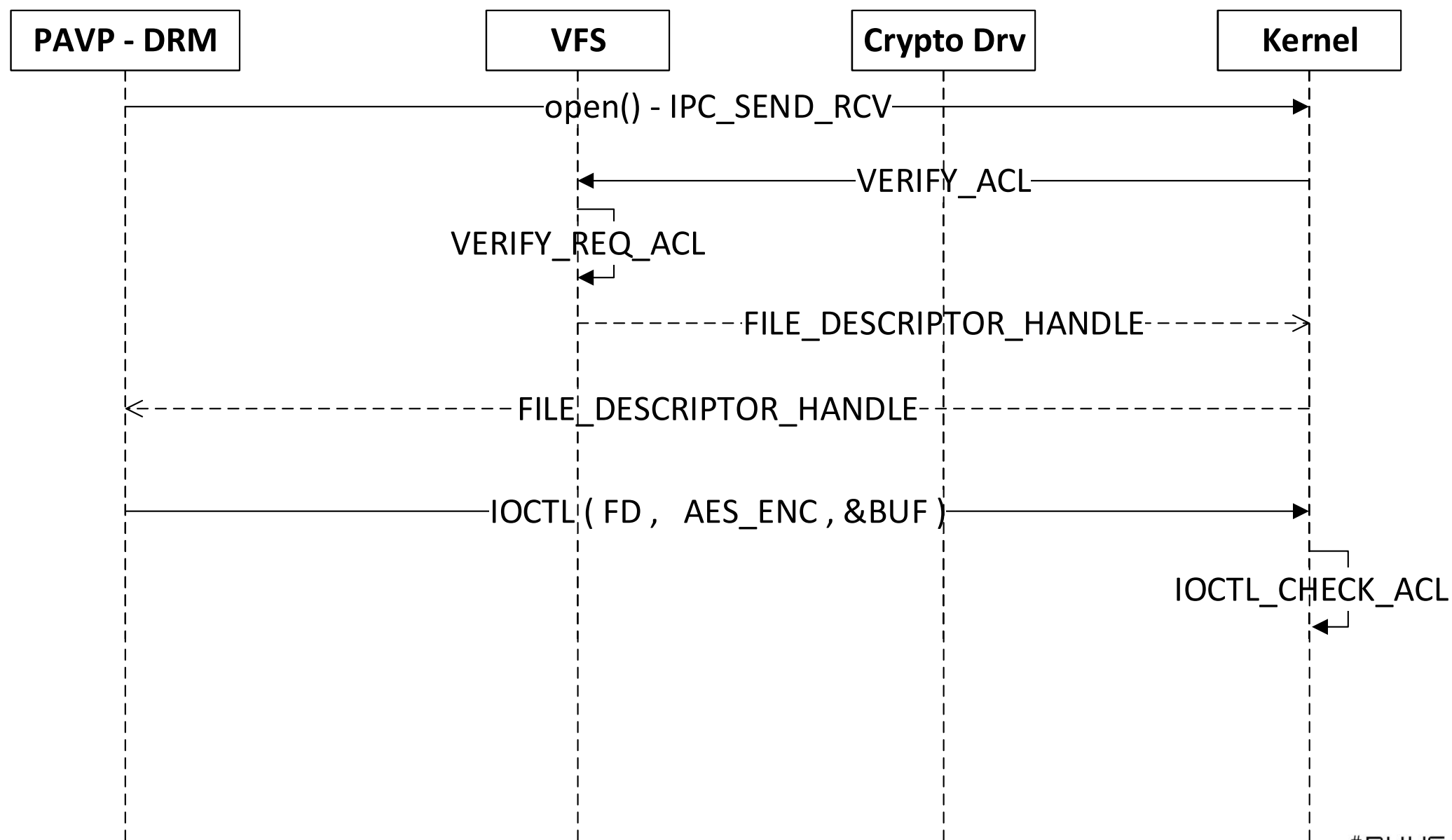
- Example of authorized IPC (Abstracted)
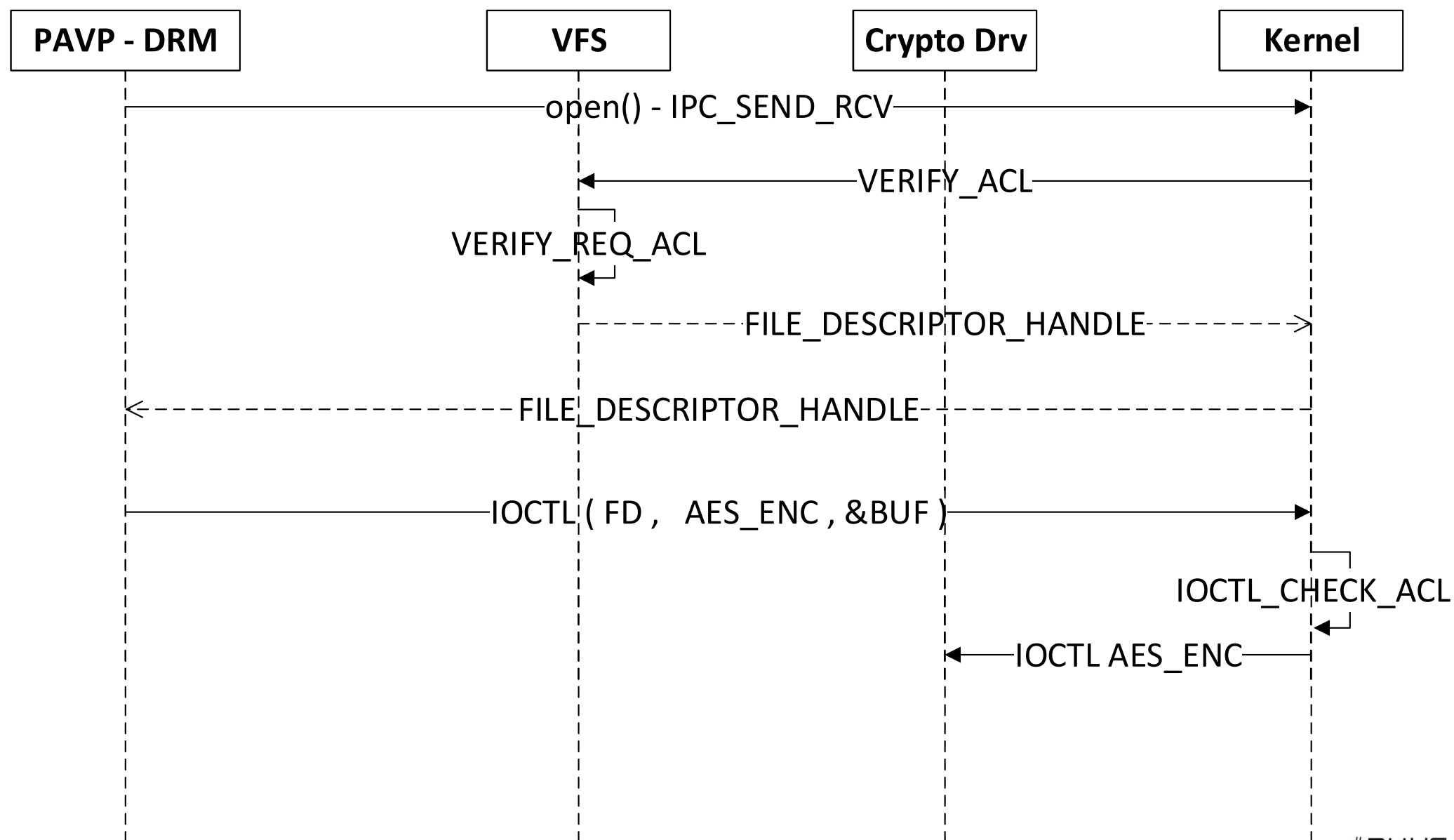
- Example of authorized IPC (Abstracted)
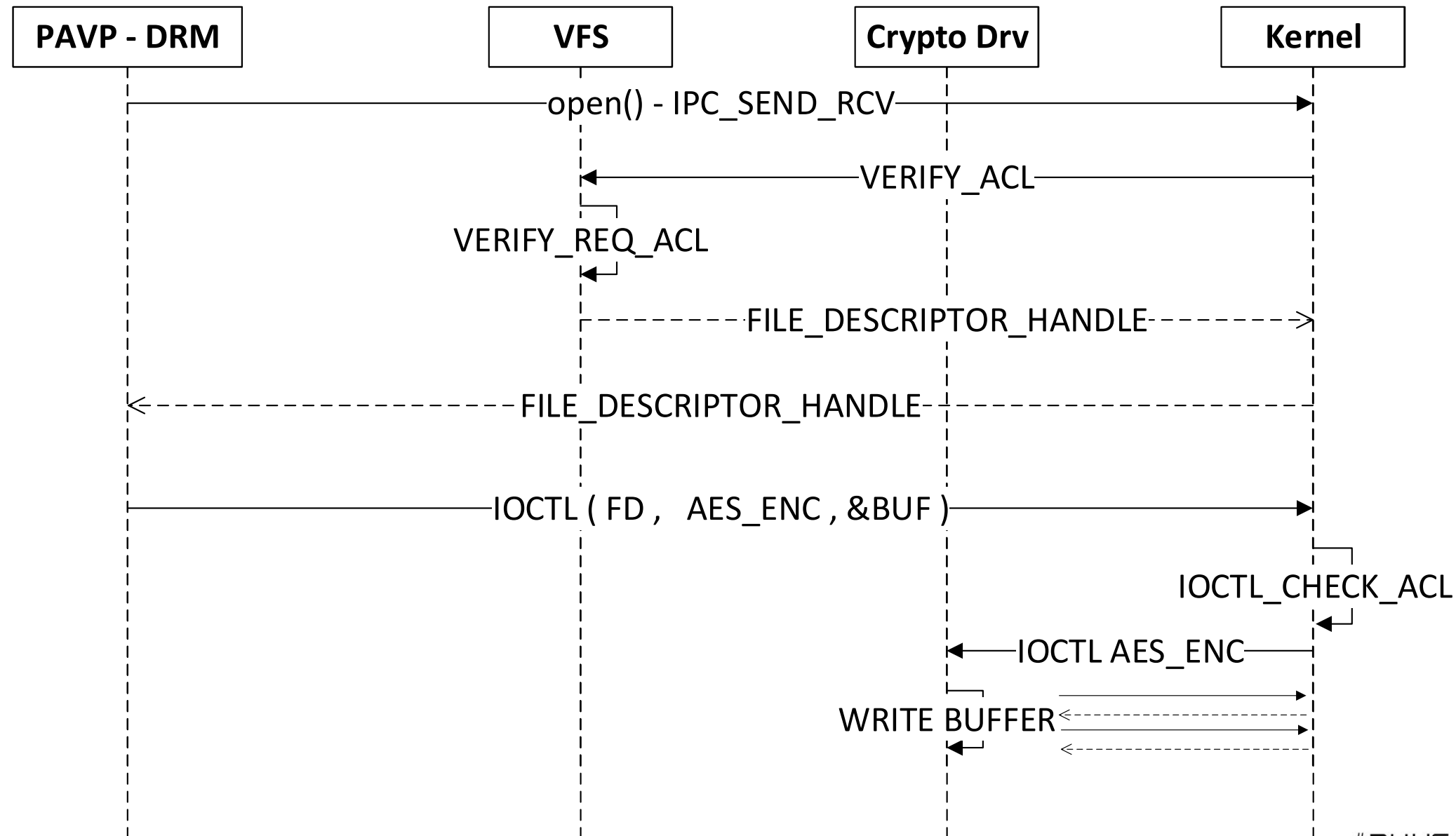
- Example of authorized IPC (Abstracted)
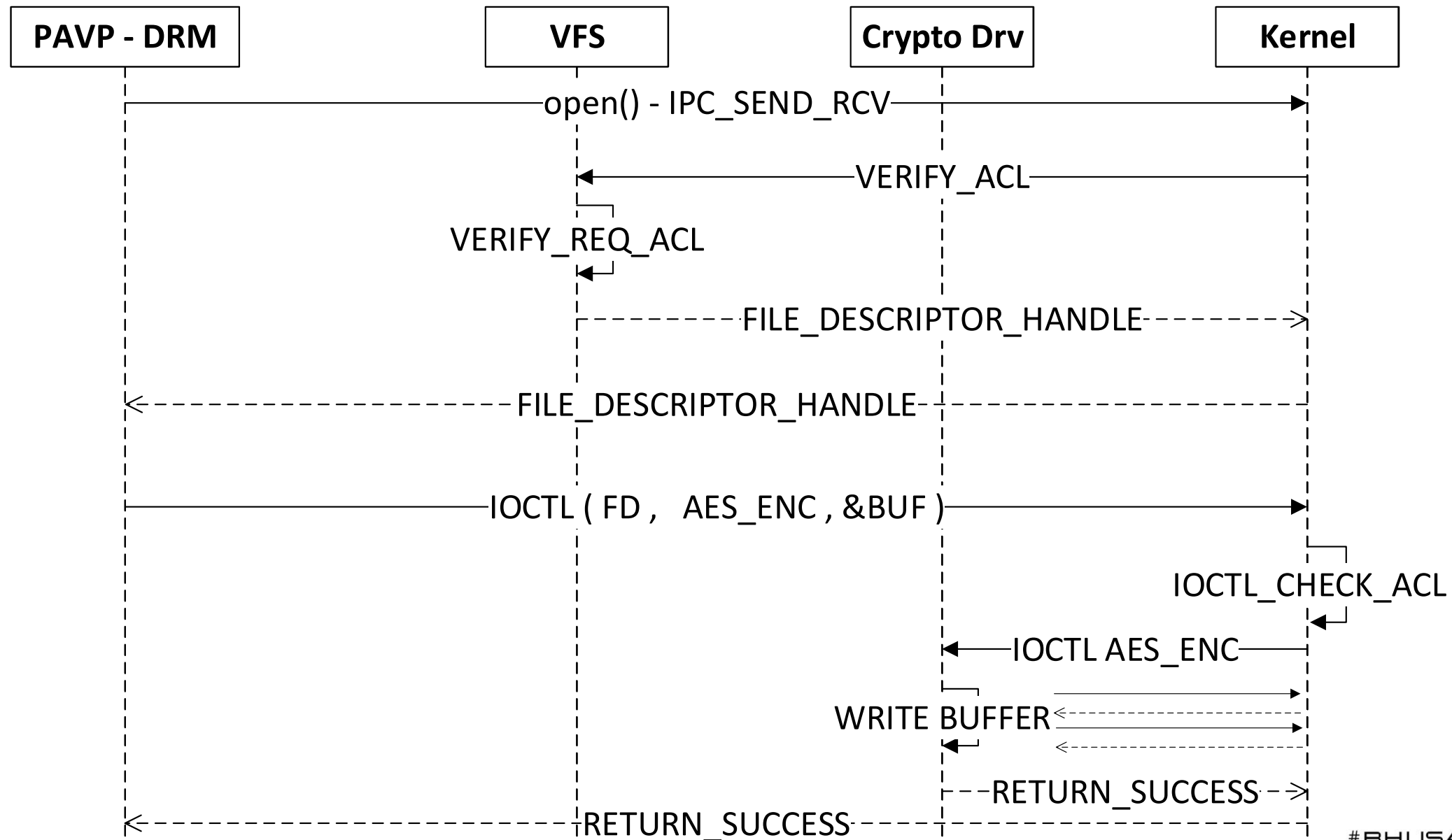
- Example of unauthorized IPC
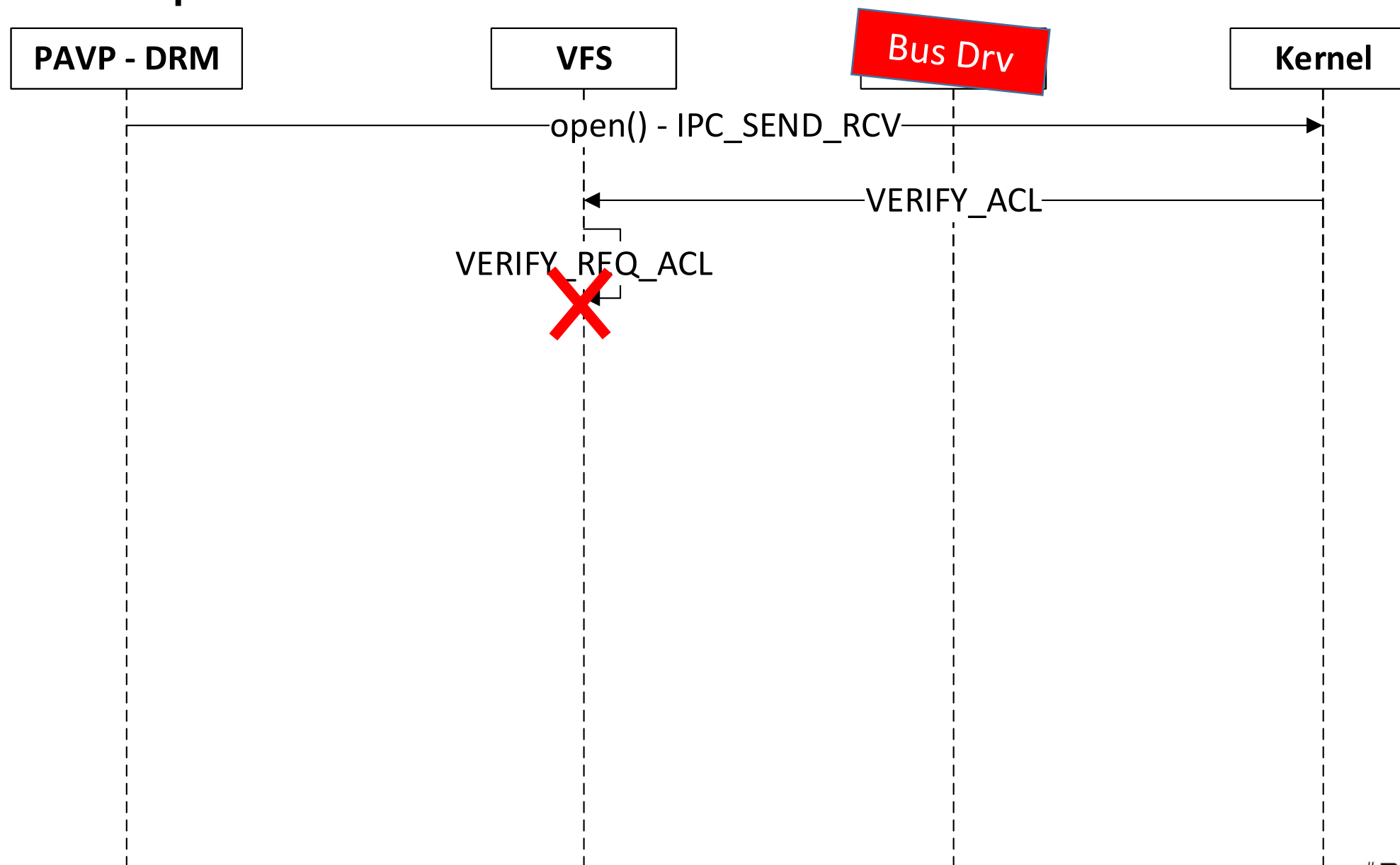
- Exploitation mitigations in Ring3

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.

- Exploitation mitigations in Ring3
    - Syslib context pointer moved to a read only page (not on stack anymore)
    - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.

grsecurity @grsecurity · Jun 26

Yeah, it seems like the XOR with retaddr was removed though after 1.21. For reference, it exists here:

marc.info/?l=stackguard&...

But then appears to be gone in all future versions:

blackhat.com/presentations/...

cs.purdue.edu/homes/xyzhang/...

(discussing v2.0.1 lacking it, and 1.21 having it)

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.

grsecurity @grsecurity · Jun 26
Yeah, it seems like the XOR with retaddr was removed though after 1.21. For

Doctor Crispistein 🤖 @CrispinCowan0 · Jun 26
"Why" was we did some analysis that said it was no longer necessary. That analysis was wrong, of course :) but I don't remember the details.

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.

  - Regular Stack-Protector:

| LOCAL VAR | Canary | OLD EBP | RETURN | ARG 1 | ARG 2 |

Linear Buffer overflow

Random

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.

  - Regular Stack-Protector:

LOCAL VAR	Canary	OLD EBP	RETURN	ARG 1	ARG 2

Linear Buffer overflow
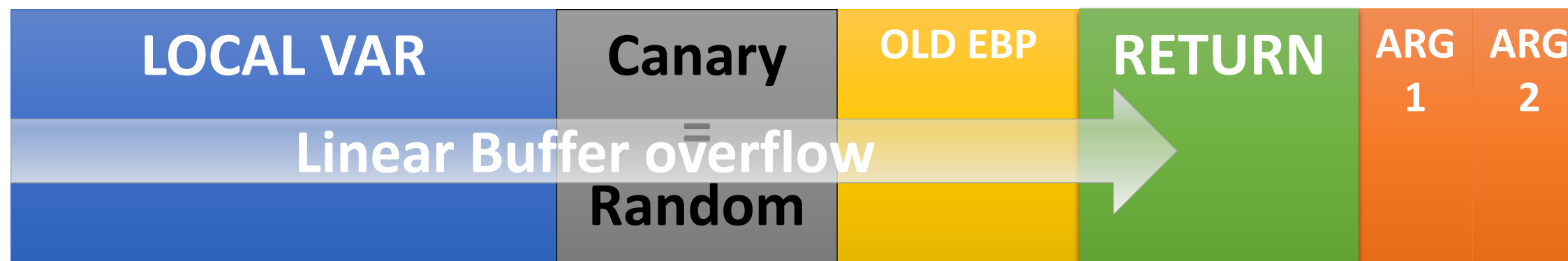
Nonlinear Write will **bypass** stack protector

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.

  - Stack-Protector XORed with Return address:

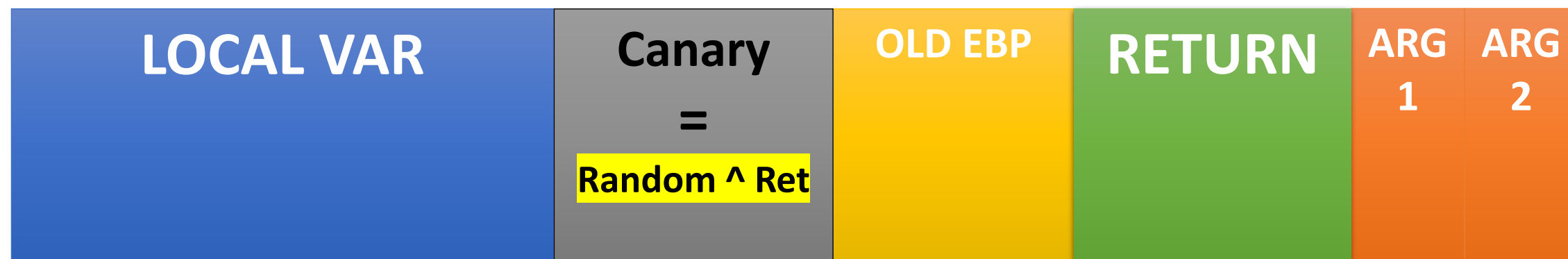| LOCAL VAR	Canary = Random ^ Ret	OLD EBP	RETURN	ARG 1	ARG 2

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
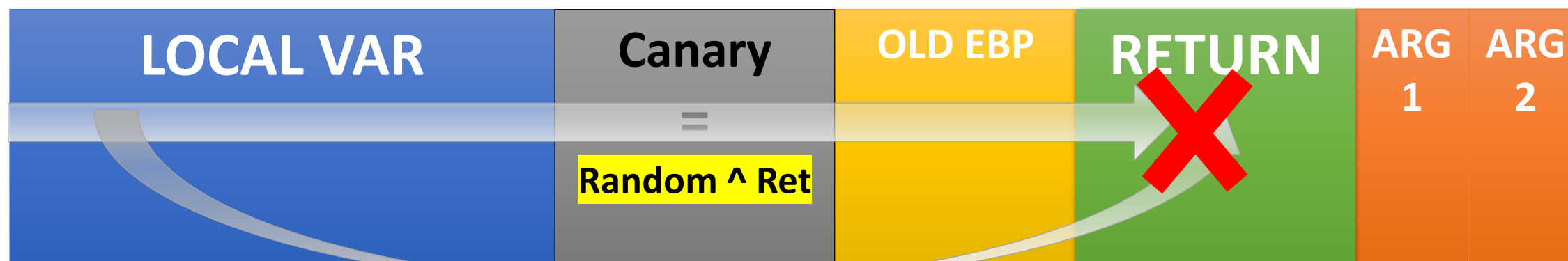
  - Stack-Protector XORed with Return address:

LOCAL VAR	Canary = Random ^ Ret	OLD EBP	RETURN ✗	ARG 1	ARG 2

- Attacker will now require to have stack/canary info leak or to leverage a data corruption (if possible)

- Exploitation mitigations in Ring3
    - Syslib context pointer moved to a read only page (not on stack anymore)
    - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
    - SW Forward Edge Control Flow Integrity

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity

COP/ROP/JOP Gadget

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity

```
00 c7 44 24 10 00 00 00 00 8b 45 08 89 44 24 08 8b
50 e8 48 ff ff ff cc cc cc cc cc cc 83 ec 0c dd 14
e8 c8 fc ff ff e8 0d 00 00 00 83 c4 0c c3 8d 54 24
e8 73 fc ff ff 52 9b d9 3c 24 74 50 66 81 3c 24 7f
74 06 d9 2d 7c e7 90 7c d9 ff 9b df e0 9e 7a 1d 83
ac e0 97 7c 00 0f 85 a2 fc ff ff ba 12 00 00 00 8d
00 e0 97 7c e9 9f fc ff ff db 2d 94 e6 90 7c d9 c9
f5 9b df e0 9e 7a f8 dd d9 d9 ff eb cd e8 05 fc ff
eb 1b a9 ff ff 0f 00 75 f2 83 7c 24 08 00 75 eb dd
db 2d 68 e0 97 7c b8 01 00 00 00 83 3d ac e0 97 7c
```

```
mov        eax, [ebp + CBFuncPtr]
```
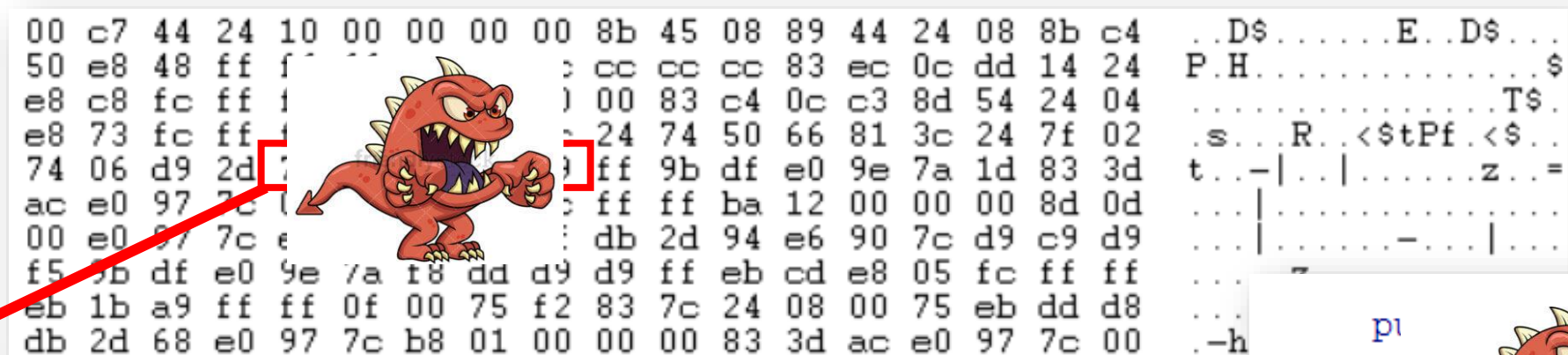
```
push       ebp
mov        ebp, esp
mov        eax, [ebp+s.pFuncs]
mov        edx, [ebp+memaccess]
mov        [ebp+s.pFuncs], edx
pop        ebp
mov        eax, [eax+1Ch]
```

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity

```
00 c7 44 24 10 00 00 00 00 8b 45 08 89 44 24 08 8b
50 e8 48 ff ff ff cc cc cc cc cc cc 83 ec 0c dd 14
e8 c8 fc ff ff e8 0d 00 00 00 83 c4 0c c3 8d 54 24
e8 73 fc ff ff 52 9b d9 3c 24 74 50 66 81 3c 24 7f
74 06 d9 2d 7c e7 90 7c d9 ff 9b df e0 9e 7a 1d 83
ac e0 97 7c 00 0f 85 a2 fc ff ff ba 12 00 00 00 8d
00 e0 97 7c e9 9f fc ff ff db 2d 94 e6 90 7c d9 c9
f5 9b df e0 9e 7a f8 dd d9 d9 ff eb cd e8 05 fc ff
eb 1b a9 ff ff 0f 00 75 f2 83 7c 24 08 00 75 eb dd
db 2d 68 e0 97 7c b8 01 00 00 00 83 3d ac e0 97 7c
```

```
mov     eax, [ebp + CBFuncPtr]
mov     ecx,  eax
mov     ecx , [ecx - 4]
```

```
endbr32
push    ebp
mov     ebp, esp
mov     eax, [ebp+s.pFuncs]
mov     edx, [ebp+memaccess]
mov     [ebp+s.pFuncs], edx
pop     ebp
mov     eax, [eax+1Ch]
```

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity


```
00 c7 44 24 10 00 00 00 00 8b 45 08 89 44 24 08 8b
50 e8 48 ff ff ff cc cc cc cc cc cc 83 ec 0c dd 14
e8 c8 fc ff ff e8 0d 00 00 00 83 c4 0c c3 8d 54 24
e8 73 fc ff ff 52 9b d9 3c 24 74 50 66 81 3c 24 7f
74 06 d9 2d 7c e7 90 7c d9 ff 9b df e0 9e 7a 1d 83
ac e0 97 7c 00 0f 85 a2 fc ff ff ba 12 00 00 00 8d
00 e0 97 7c e9 9f fc ff ff db 2d 94 e6 90 7c d9 c9
f5 9b df e0 9e 7a f8 dd d9 d9 ff eb cd e8 05 fc ff
eb 1b a9 ff ff 0f 00 75 f2 83 7c 24 08 00 75 eb dd
db 2d 68 e0 97 7c b8 01 00 00 00 83 3d ac e0 97 7c
```

```
mov      eax, [ebp + CBFuncPtr]
mov      ecx,  eax
mov      ecx , [ecx - 4]
dec      ecx
#compare with endbr32-1 to create "cmp" without endbr32 bytecode
cmp      ecx, 0FB1E0FF2h   #endbr32 = f3 0f 1e fb
```

```
endbr32
push     ebp
mov      ebp, esp
mov      eax, [ebp+s.pFuncs]
mov      edx, [ebp+memaccess]
mov      [ebp+s.pFuncs], edx
pop      ebp
mov      eax, [eax+1Ch]
```

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity


```
mov      eax, [ebp + CBFuncPtr]
mov      ecx,  eax
mov      ecx , [ecx - 4]
dec      ecx
#compare with endbr32-1 to create "cmp" without endbr32 bytecode
cmp      ecx, 0FB1E0FF2h  #endbr32 = f3 0f 1e fb
jz       .L_valid_function_pointer
call     __CFI_FAIL__
.L_valid_function_pointer:
call     eax
```

```
00 c7 44 24 10 00 00 00 00 8b 45 08 89 44 24 08 8b
50 e8 48 ff ff ff cc cc cc cc cc cc 83 ec 0c dd 14
e8 c8 fc ff ff e8 0d 00 00 00 83 c4 0c c3 8d 54 24
e8 73 fc ff ff 52 9b d9 3c 24 74 50 66 81 3c 24 7f
74 06 d9 2d 7c e7 90 7c d9 ff 9b df e0 9e 7a 1d 83
ac e0 97 7c 00 0f 85 a2 fc ff ff ba 12 00 00 00 8d
00 e0 97 7c e9 9f fc ff ff db 2d 94 e6 90 7c d9 c9
f5 9b df e0 9e 7a f8 dd d9 d9 ff eb cd e8 05 fc ff
eb 1b a9 ff ff 0f 00 75 f2 83 7c 24 08 00 75 eb dd
db 2d 68 e0 97 7c b8 01 00 00 00 83 3d ac e0 97 7c
```

```
endbr32
push     ebp
mov      ebp, esp
mov      eax, [ebp+s.pFuncs]
mov      edx, [ebp+memaccess]
mov      [ebp+s.pFuncs], edx
pop      ebp
mov      eax, [eax+1Ch]
```

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity
  - Heap protections

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity
  - Heap protections
    - Double free protection
    - Malloc of zero size return NULL
    - Cookie protection enforced during free of an allocated\busy chunk
      - A Marker surround every Busy Block
      - Value of random "Cookie" field in the Marker is compared with the original Cookie value. Mismatch is handled as an overflow attack (or bug).

- Exploitation mitigations in Ring3
  - Syslib context pointer moved to a read only page (not on stack anymore)
  - Return Control Flow Integrity via modified Stack Canary "XOR-RET-ALL" on the majority of the Ring3 functions.
  - SW Forward Edge Control Flow Integrity
  - Heap protections
  - Data execution prevention

Applying Security Development Lifecycle (SDL) through the CSME development phases

- Security Architecture and Design Review
  - Threat analysis
  - Challenging FW design results into product changes
- Security Code Review
  - Manual and Static Code Analysis (SCA) tools
- Penetration testing
  - Manual
  - Automation

- Using latest industry techniques on silicon

- Using latest industry techniques on silicon
  - Address Sanitization

- Using latest industry techniques on silicon
  - Address Sanitization
    - Doesn't work out of the box since requires glibc
    - Sanitizer requires 8 bytes aligned memory and CSME a 4 bytes aligned memory
    - Sanitizer write to a "shadow" memory at a fixed address of "0x20000000"
    - Making the code too big won't fit into flash or won't fit into SRAM

- ## Using latest industry techniques on silicon
  - ### Address Sanitization
    - ~~Doesn't work out of the box since requires glibc.~~
      - Fixed it by creating stub functions that are missing and implement them
    - ~~Sanitizer requires 8 bytes aligned memory and CSME a 4 bytes aligned memory~~
      - Each time we enter a "error function" make sure is it in a 4 byte aligned memory and validate that address with the "shadow" if it's an issue or not
      - Using: "-fsanitize-recover=address" so calling code path won't change
    - ~~Sanitizer write to a "shadow" memory at a fixed address of "0x20000000"~~
      - Patch GCC to make it accept "-fsanitize=kernel-address" by removing "SANITIZE_KERNEL_ADDRESS" in "opts-global.c"
      - Was asked many times but none done it as a feature in GCC:
        - https://groups.google.com/forum/#!topic/address-sanitizer/ZLI4un1NyoE
    - ~~Making the code too big won't fit into flash or won't fit into SRAM~~
      - Apply only on a single process and not on the entire system at once.

- Using latest industry techniques on silicon
  - Address Sanitization ✓

- Using latest industry techniques on silicon
  - Address Sanitization ✅
  - Fuzzing with Coverage guided

- Using latest industry techniques on silicon
  - Address Sanitization ✓
  - Fuzzing with Coverage guided
    - Based on AFL Fuzzer logic

- Using latest industry techniques on silicon
  - Address Sanitization ✓
  - Fuzzing with Coverage guided
    - Based on AFL Fuzzer logic

    Issue #1: BitMap size

- Using latest industry techniques on silicon
  - Address Sanitization ✅
  - Fuzzing with Coverage guided
    - Based on AFL Fuzzer logic

    Issue #1: BitMap size

```
/* Map size for the traced binary (2^MAP_SIZE_POW2). Must be greater than
   2; you probably want to keep it under 18 or so for performance reasons
   (adjusting AFL_INST_RATIO when compiling is probably a better way to solve
   problems with complex programs). You need to recompile the target binary
   after changing this - otherwise, SEGVs may ensue. */

#define MAP_SIZE_POW2          16
```

- Using latest industry techniques on silicon
  - Address Sanitization ✓
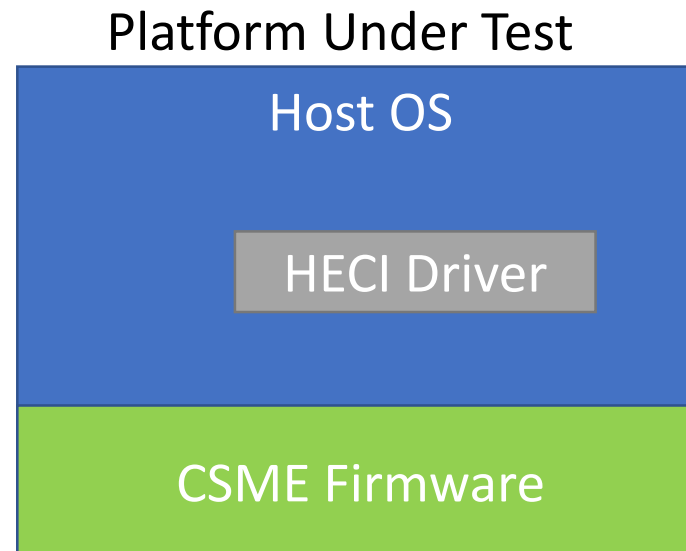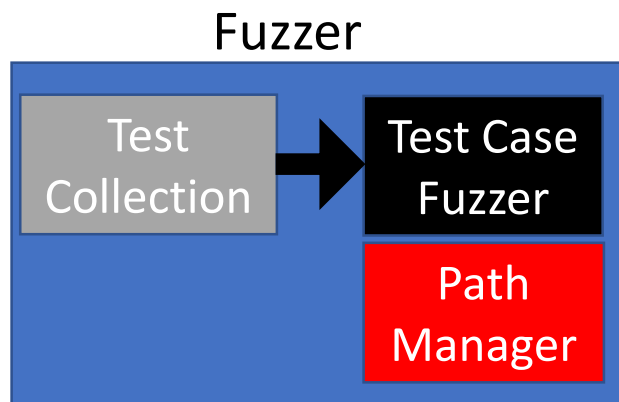 - Fuzzing with Coverage guided
    - Based on AFL Fuzzer logic

    Issue #2: Memory pipe for getting test feedback

- Using latest industry techniques on silicon
  - Address Sanitization ✅
  - Fuzzing with Coverage guided
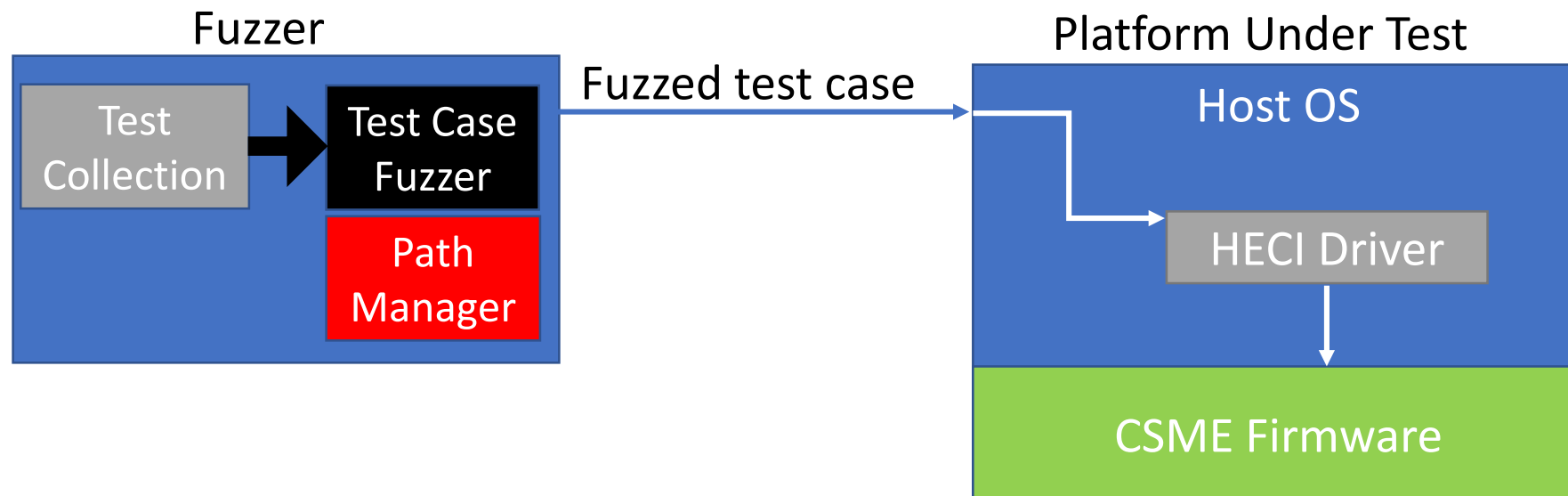   - Based on AFL Fuzzer logic

    Issue #2: Memory pipe for getting test feedback
      - Easy to solve by calling a test firmware API (not exist in production) to get the internal array that hold all feedback
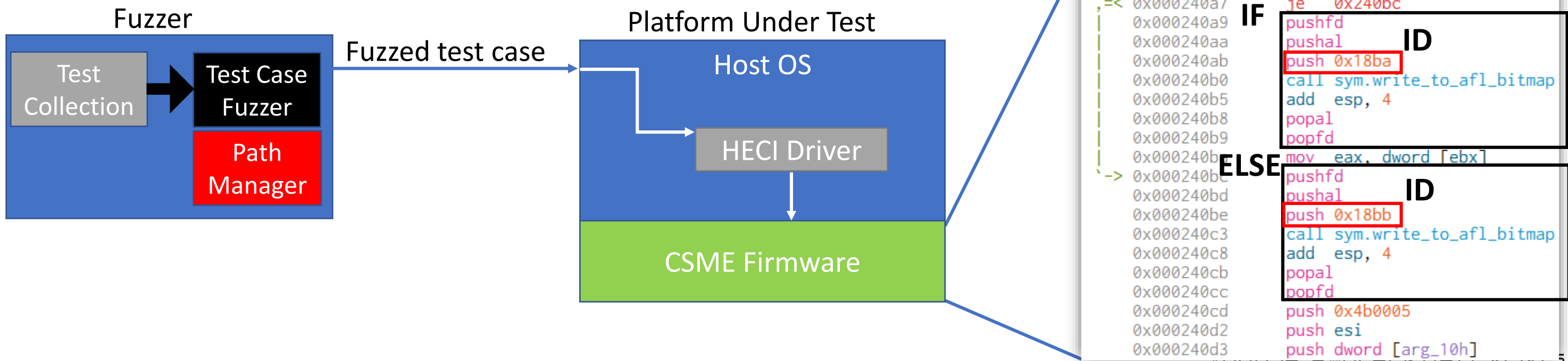   - Modify AFL instrumentation to set the global BITMAP array inside of the FW

- Using latest industry techniques on silicon
  - Address Sanitization ✅
  - Fuzzing with Coverage guided
    - Based on AFL Fuzzer logic

- Using latest industry techniques on silicon
  - Address Sanitization ✅
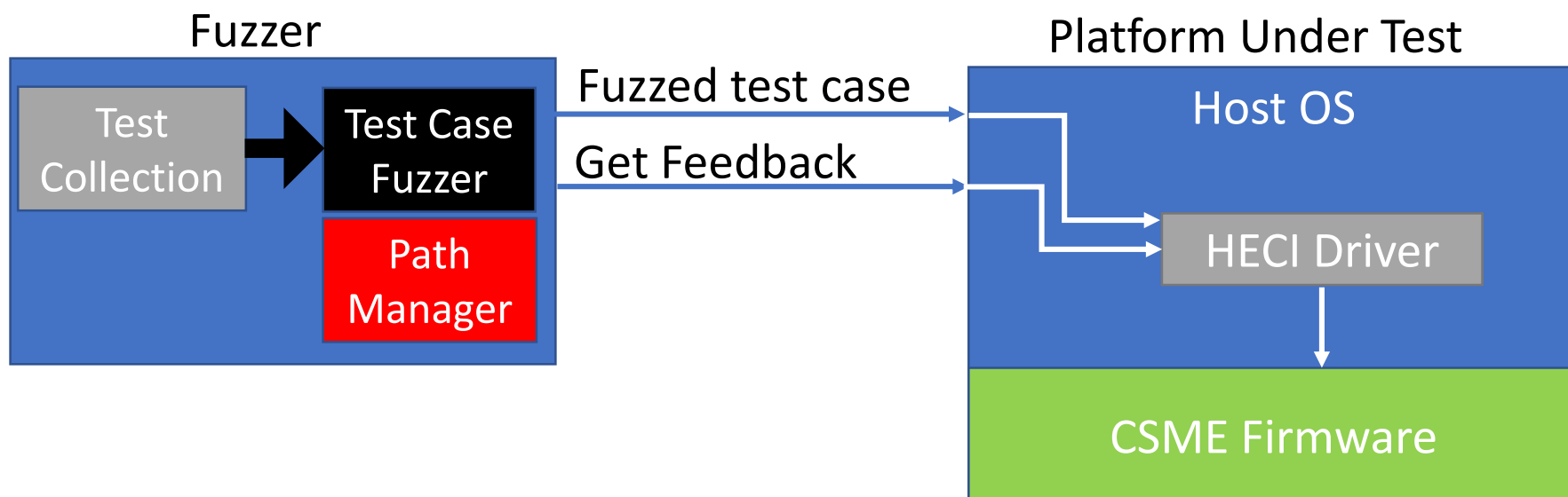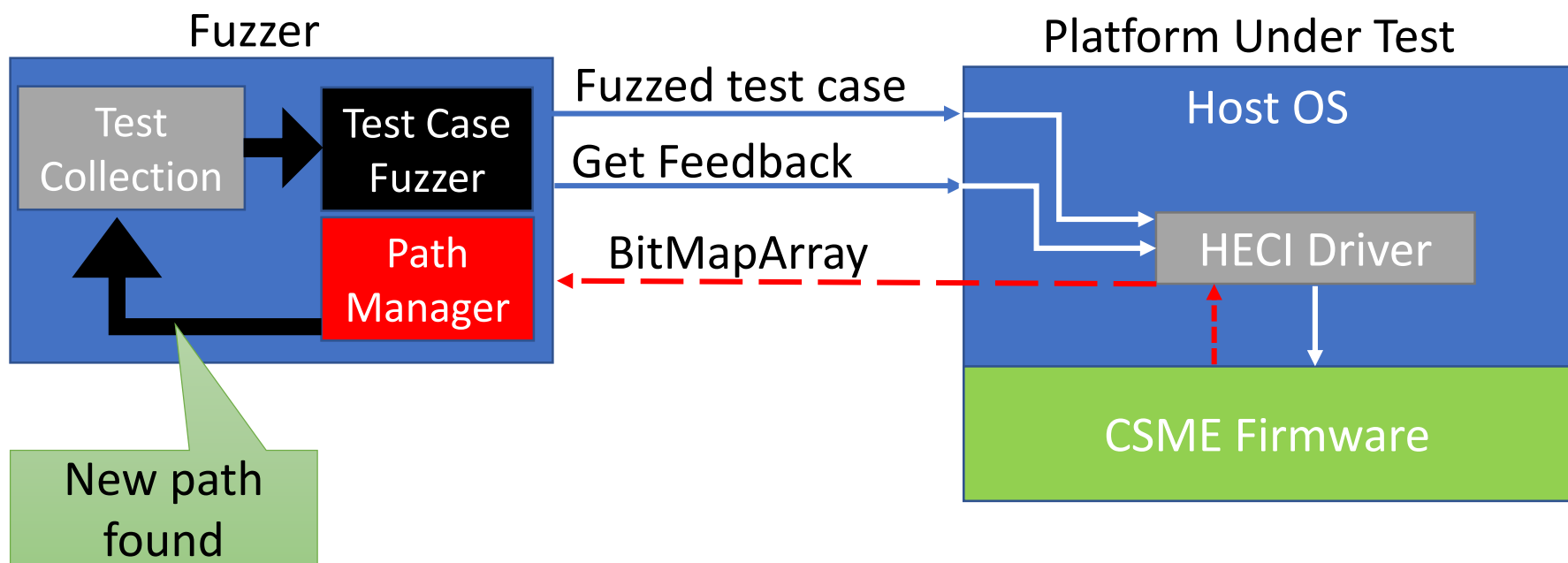 - Fuzzing with Coverage guided
    - Based on AFL Fuzzer logic

Instrumented Firmware

Fuzzer

Platform Under Test

Fuzzed test case

Test Collection → Test Case Fuzzer

Path Manager

Host OS

HECI Driver

CSME Firmware

```
0x0002409f    mov   ebx, dword [arg_18h] ;
0x000240a2    mov   esi, dword [arg_ch] ; [
0x000240a5    test  ebx, ebx
0x000240a7    je    0x240bc
0x000240a9    IF    pushfd
0x000240aa          pushal              ID
0x000240ab          push  0x18ba
0x000240b0          call  sym.write_to_afl_bitmap
0x000240b5          add   esp, 4
0x000240b8          popal
0x000240b9          popfd
0x000240b    ELSE   mov   eax, dword [ebx]
0x000240bc          pushfd
0x000240bd          pushal              ID
0x000240be          push  0x18bb
0x000240c3          call  sym.write_to_afl_bitmap
0x000240c8          add   esp, 4
0x000240cb          popal
0x000240cc          popfd
0x000240cd          push  0x4b0005
0x000240d2          push  esi
0x000240d3          push  dword [arg_10h]
```

- Using latest industry techniques on silicon
  - Address Sanitization ✅
  - Fuzzing with Coverage guided
    - Based on AFL Fuzzer logic

Instrumented Firmware

Fuzzer

Platform Under Test

- Using latest industry techniques on silicon
  - Address Sanitization ✅
  - Fuzzing with Coverage guided
    - Based on AFL Fuzzer logic

Instrumented Firmware

- Using latest industry techniques on silicon
  - Address Sanitization ✓
  - Fuzzing with Coverage guided ✓

- Architecture & Boot flow
- OS Security Principles & Internals
- Hardening & Mitigations
- **Pre & Post Manufacturing**
- Update & Recoverability
- Wrap-up

- Features are configurable by manufacturers
  - Manageability support – corporate / consumer
  - HW Anti-rollback support
  - Manufacturer public key for secure micro-code loading and Intel Boot Guard
  - Intel Boot Guard enable/disable & policy
  - PTT enable/disable

- End Of Manufacturing (EOM)
  - Required by manufacturers before shipping platforms to end-users
  - Write and lock manufacturers' settings into FPF and CSME data partition in SPI flash
  - Close SPI flash descriptor – SPI controller enforces access control on BIOS, CSME and other SPI regions

Some CSME features can still be configured after EOM by end-users

- Manageability can be configured in BIOS menus
    - AMT out of band network interface enable/disable
    - AMT USB provisioning enable/disable
    - AMT Host Based Provisioning enable/disable
   - AMT redirection enable/disable

- Architecture & Boot flow
- OS Security Principles & Internals
- Hardening & Mitigations
- Pre & Post Manufacturing
- Update & Recoverability
- Wrap-up

CSME FW verifies digital signature and version of new CSME FW image before updating it in SPI flash on end-user system

- Two levels of CSME FW anti-rollback supported in CSME 12
    1. SW rollback to old CSME FW is prevented using Version Control Number (VCN)
    2. Physical rollback is prevented using Anti-Rollback (ARB) SVN
        - ARB SVN is kept in field programmable fuse (FPF)
        - Require manufacturer support

- Once FW update is done to a higher TCB SVN, CSME will perform data migration and initiate the re-creation of attestation keys (EPID and PTT Endorsement Key)
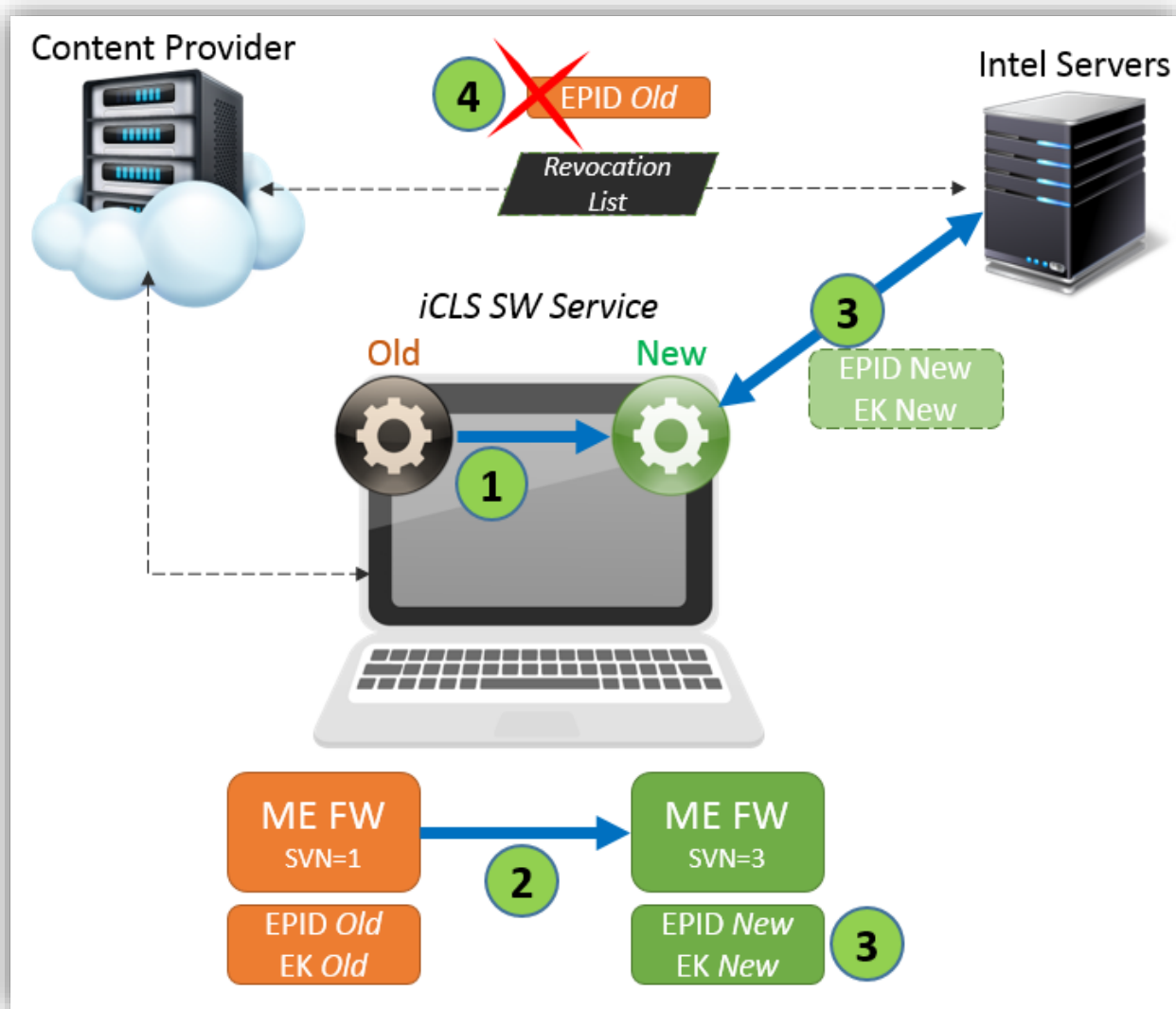
1. If not latest iCLS (Intel Capability Licensing Service) SW service is already used, update SW.

2. Manufacturers update to new CSME FW with **higher SVN**. At next boot, CSME FW performs CSME data migration from previous CSME storage key to new one derived by ROM.

3. Intel iCLS SW service connects securely using Intel SIGMA protocol over internet to Intel backend servers to complete TCB recovery and retrieve new EPID key and Intel certificate for new PTT Endorsement Key (TPM EK)

4. At some point, Intel revokes EPID keys and PTT EK (i.e. publish CRL on Intel server). Once revocation is done, Content Providers can halt streaming content to non-updated systems

- Architecture & Boot flow
- OS Security Principles & Internals
- Hardening & Mitigations
- Pre & Post Manufacturing
- Update & Recoverability
- Wrap-up

- Secure design with defense In-depth
  - Secure boot and execution enforced by minimal TCB
  - Least privileges and process isolation
  - Exploitation mitigations

- Secure Update & Recovery
  - Secure FW update
  - FW and HW Anti-rollback
  - Data migration with online renewal of attestation keys (TCB Recovery)

- Evaluating Future Enhancements
  - Further reduce privileges
  - Adding ASLR support given CSME OS memory limitations
  - HW Control-Flow Enforcement Technology (CET) support in CSME CPU

Special thanks to CSME architecture, development, validation and security research teams for their contribution to this presentation