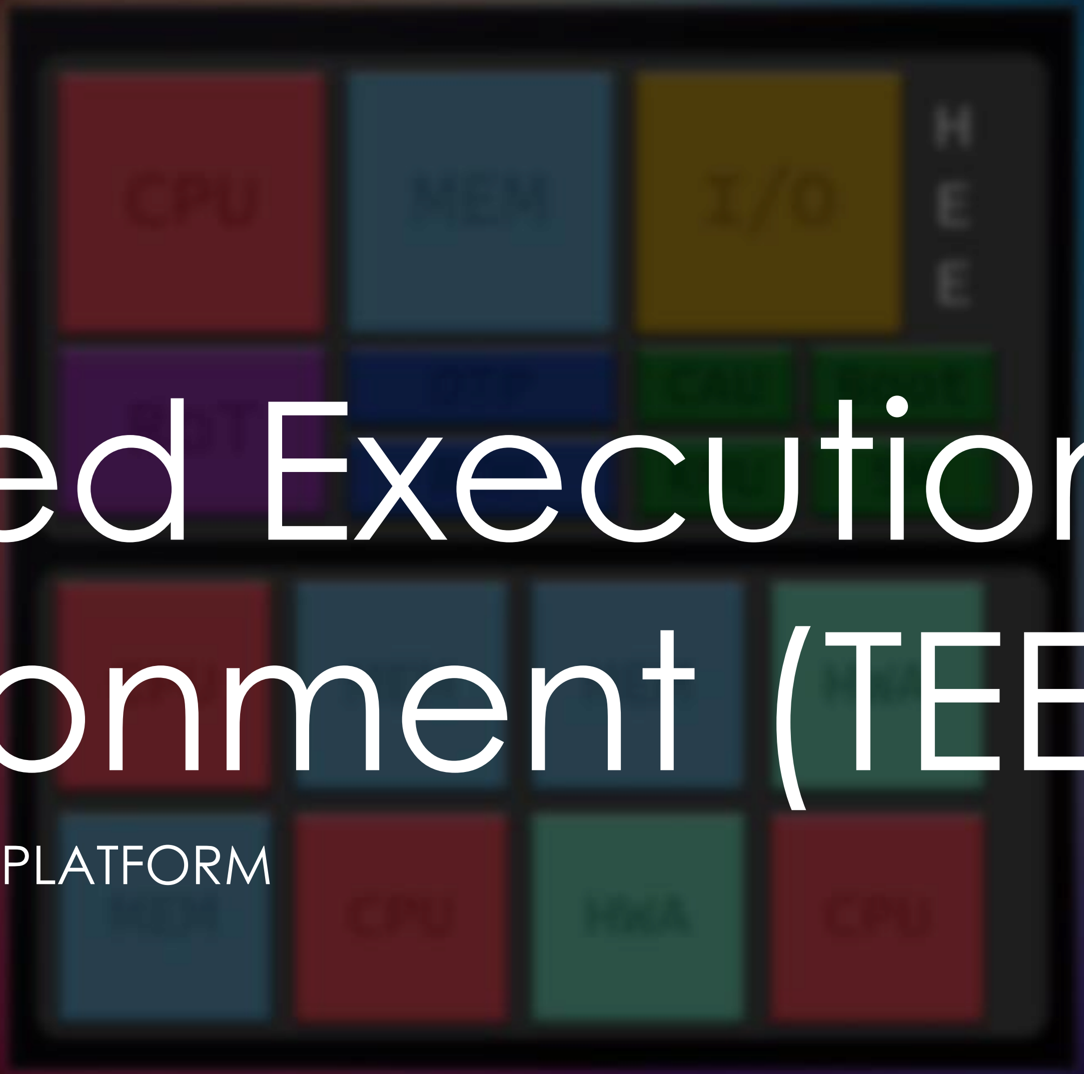


# Trusted Execution Environment (TEE)

IN THE RISC-V ESP PLATFORM

# Outline

## Introduction to TEE

- Exponentially increasing number of heterogeneous interconnected devices to exploit an increasing amount of informations.
- Devices rely on System-on-Chip (SoC) with specialized functions realized through Hardware Accelerators (HA).
- Security is fundamental for SoC used in critical system (such as aircrafts, banks, medical devies, …
- TEE can provide protection of both run-time states and assets.
- Need of fast&cheap design: RISC-V ESP Platform for fully scalable tile-based architecture with fast prototyping on FPGA.

## System-on-Chip Security

*« An **attack towards a system** is an attempt from a malicious user to extract protected information as well as modifying the behavior of the system itself or stopping it at all. »*

- TEE should theoretically be able to stop all kinds of attack, both physical and software.
- Ensuring security on SoC is challenging because of a constant trade-off with performances and increasingly sophisticated attacks.

## Software and Physical Attacks

- RTL Bugs
- Hardware Trojans (HT)
- Logic-locking attacks
- Electromagnetic Fault Injection (EMFI)
- Covert channels
- Physical access attacks
- Buffer overflow attacks
- Side-channel attacks
- … and more!!

## Attacks Countermeasures

- Trusted computing
- Cryptographic engines
- Memory Protection
- Oblivious RAM (ORAM)
- Side-channel attacks prevention

## Outline

- **SoC Security: Introduction and threat model**
- **RISC-V ESP Tile-based Architecture**
- **Trusted Execution Environment (TEE) implementation:**
  - Overall Architecture overview
  - Separation Kernel and TEE Definition
  - The Root of Trust (RoT)
  - System Boot Sequence
  - Memory Protection strategies
  - Encryption in TEE
  - Other protection techinques

## Introduction to RISC-V ESP Platform

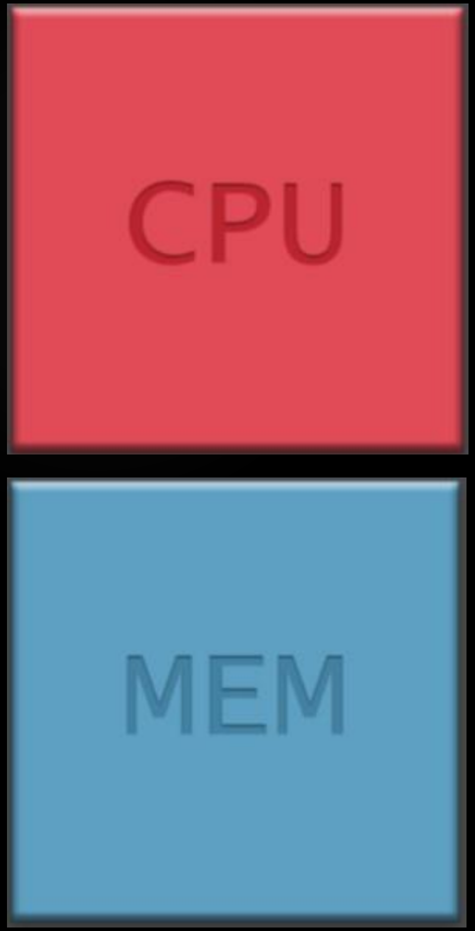
*“ESP is an open-source research platform for heterogeneous SoC design. The platform combines a modular tile-based architecture with a variety of application-oriented flows for the design and optimization of accelerators. The ESP architecture is highly scalable and strikes a balance between regularity and specialization. The companion methodology raises the level of abstraction to system-level design and enables an automated flow from software and hardware development to full-system prototyping on FPGA. For application developers, ESP offers domain-specific automated solutions to synthesize new accelerators for their software and to map complex workloads onto the SoC architecture. For hardware engineers, ESP offers automated solutions to integrate their accelerator designs into the complete SoC.”*

# ESP Architecture

- It is structured as a heterogeneous grid filled with a mix of tiles chosen depending on the target application.
- Each tile is implented in a modular socket (aka shell) which decouples it from the generic interconnection system and provide L2 cache, proxies, etc…
- Tiles are connected through a Multiplane Network-on-Chip (NoC), which features buses that receive calls from the masters (cores and accelerators) and forward them to target slaves (memory, I/O ports, accelerators). Buses feature buffer queue and multi-level communications to provide enough bandwidth.
- Same importance for both cores and specialized functions! (System centric instead of Core-centric architecture).

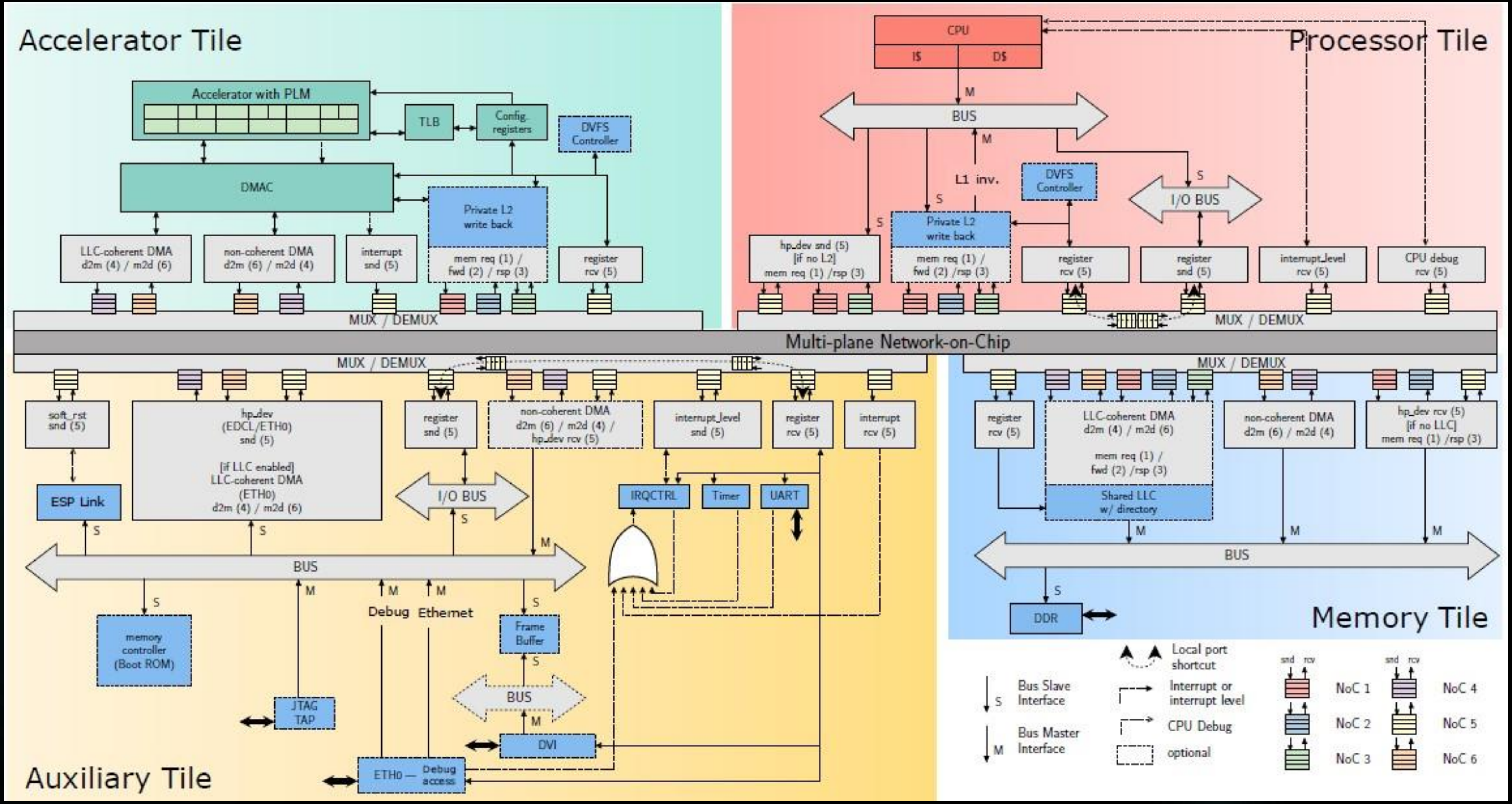
## Tiles (1)

- **Processor Tile**: contain a processore capable of running Linux with its own L1 cache. It communicates with a local bus and it is agnostic of the rest of the system. Communicates through an AXI interface. Any request is forwarded through the socket to the NoC.
- **Memory Tile**: contain a channel to external DRAM and all the necessary hardware logic for partitioning of addresable memory space which must be completely transparent to software.



## Tiles (2)

- **Hardware Accelerator Tile**: contain a specialized function hardware implemented, that is executed independently from the processor while exchanging large datasets. It must be designed to deal with latency-insensitive load/store ports, configurations signals, acc_done interrupt signal. It also features DMA and P2P communications to bypass requests to processor tiles. In conclusion, it also have a Private Local Memory (PLM)
- **Auxiliary I/O Tile**: hosts all shared peripherals except from memory. It feature Ethernet port, UART, Debug, Digital Video Interface (as well as frame-buffer memory dedicated to video output).


Accelerator Tile
Accelerator with PLM
TLB
Config. registers
DVFS Controller
DMAC
Private L2 write back
LLC-coherent DMA d2m (4) / m2d (6)
non-coherent DMA d2m (6) / m2d (4)
interrupt snd (5)
mem req (1) / fwd (2) / rsp (3)
register rcv (5)
MUX / DEMUX
Processor Tile
CPU
I$
D$
M
BUS
L1 inv.
S
DVFS Controller
I/O BUS
Private L2 write back
hp_dev snd (5) [if no L2] mem req (1) /rsp (3)
mem req (1) / fwd (2) /rsp (3)
register rcv (5)
register snd (5)
interrupt_level rcv (5)
CPU debug rcv (5)
Multi-plane Network-on-Chip
soft_rst snd (5)
hp_dev (EDCL/ETH0) snd (5)
[if LLC enabled] LLC-coherent DMA (ETH0) d2m (4) / m2d (6)
register snd (5)
non-coherent DMA d2m (6) / m2d (4) / hp_dev rcv (5)
interrupt_level snd (5)
register rcv (5)
interrupt rcv (5)
ESP Link
I/O BUS
IRQCTRL
Timer
UART
BUS
Debug
Ethernet
memory controller (Boot ROM)
Frame Buffer
JTAG TAP
DVI
ETH0 — Debug access
Auxiliary Tile
register rcv (5)
LLC-coherent DMA d2m (4) / m2d (6)
mem req (1) / fwd (2) /rsp (3)
Shared LLC w/ directory
non-coherent DMA d2m (6) / m2d (4)
hp_dev rcv (5) [if no LLC] mem req (1) /rsp (3)
DDR
Memory Tile
Local port shortcut
Bus Slave Interface
Bus Master Interface
Interrupt or interrupt level
CPU Debug
optional
snd rcv
NoC 1
NoC 2
NoC 3
NoC 4
NoC 5
NoC 6

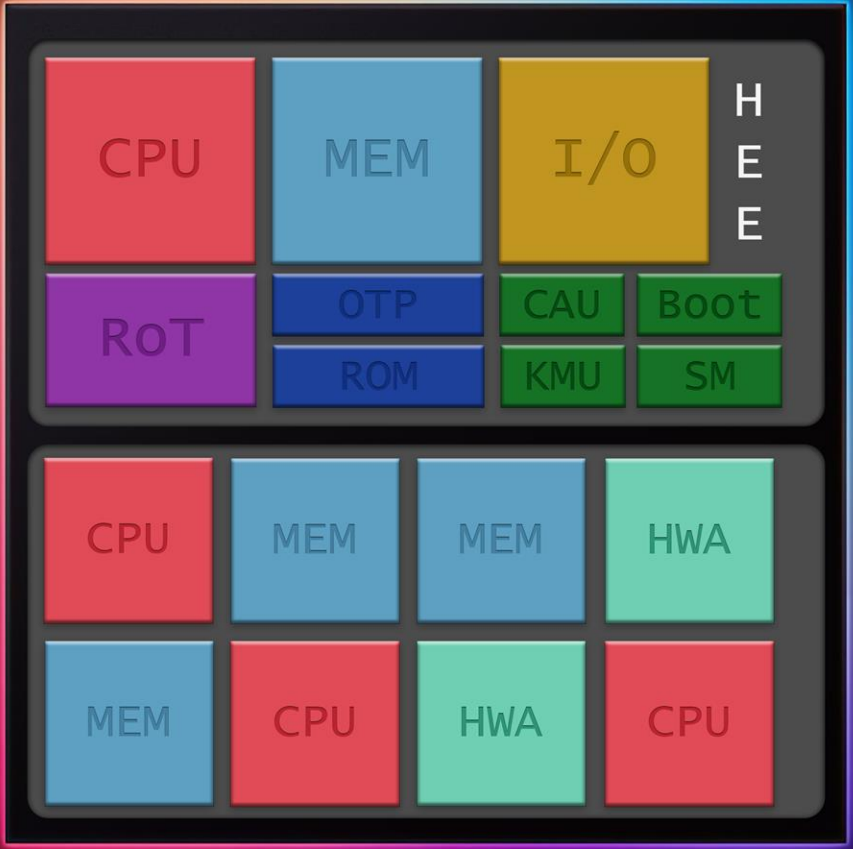
# Outline

- **SoC Security: Introduction and threat model**
- **RISC-V ESP Tile-based Architecture**
- **Trusted Execution Environment (TEE) implementation:**
  - Overall Architecture overview
  - Separation Kernel and TEE Definition
  - The Root of Trust (RoT)
  - System Boot Sequence
  - Memory Protection strategies
  - Encryption in TEE
  - Other protection techinques

# Introduction to TEE

- A **Trusted Execution Environment (TEE)** is an execution environment which protect both its run-time states and its stored assets, hence the need of isolation and secure storage.
- A TEE must guarantee:
  - Authenticity of executed code
  - Integrity of run-time states
  - Confidentiality of its code, data and run-time states on a persistent memory
  - Remote attestation of trustworthiness for third-parties
  - Securely Updtable Content
  - Resistance to all software/physical attacks on main memory and backdoor security flaws

# Overall system architecture



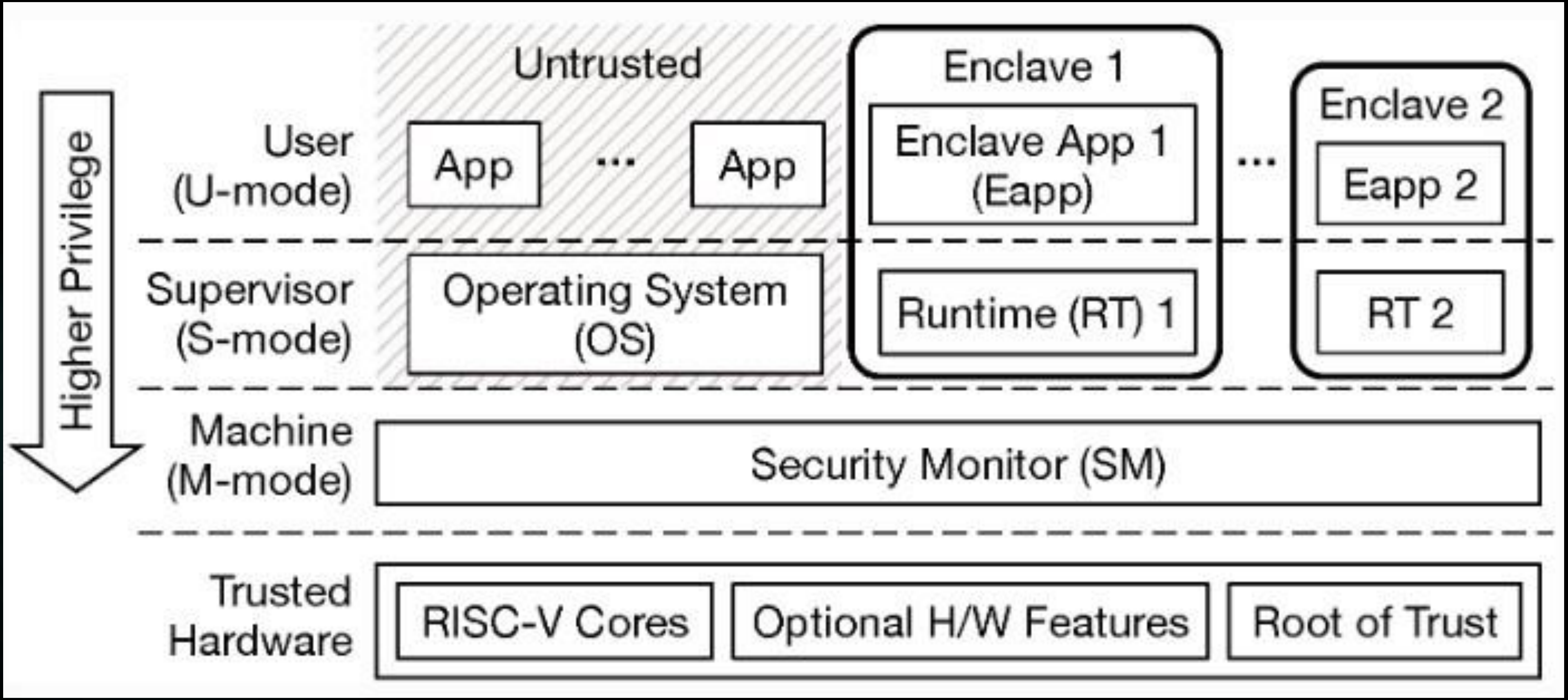
## Separation Kernel

*« The **Separation Kernel** is a foundation component of the TEE. It is the element that assures the property of isolated execution. The separation kernel [...] is a security kernel used to simulate a distributed system. Its main design purpose is to enable the coexistence of different systems requiring different levels of security on the same platform. Basically, it divides the system into several partitions, and guarantees a strong isolation between them, except for the carefully controlled interface for inter-partition communication. »*

- *The separation kernel must provide:*
  - *Data Separation (a partition cannot read or modify the data of another partition).*
  - *Temporal Separation (shared resources should not leak informations into other partitions).*
  - *Control of information flow (communications between partitions can occurr only when allowed).*
  - *Fault Isolation (a security breach in one partition should not spread into other partitions).*

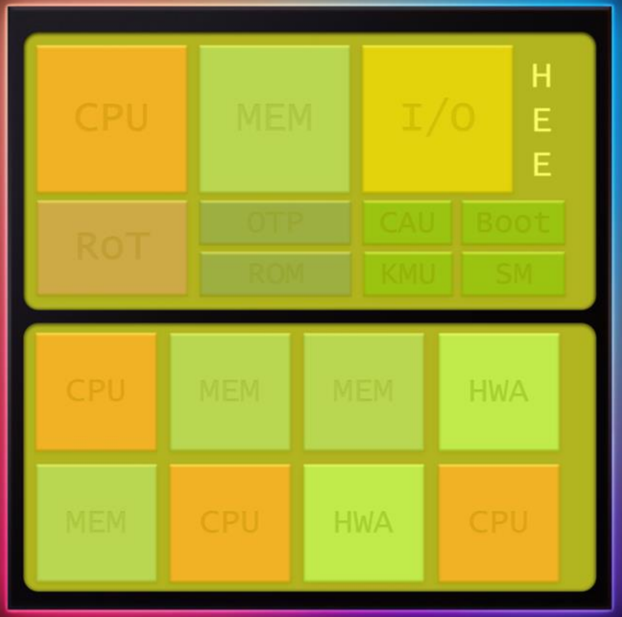
## Definition of TEE

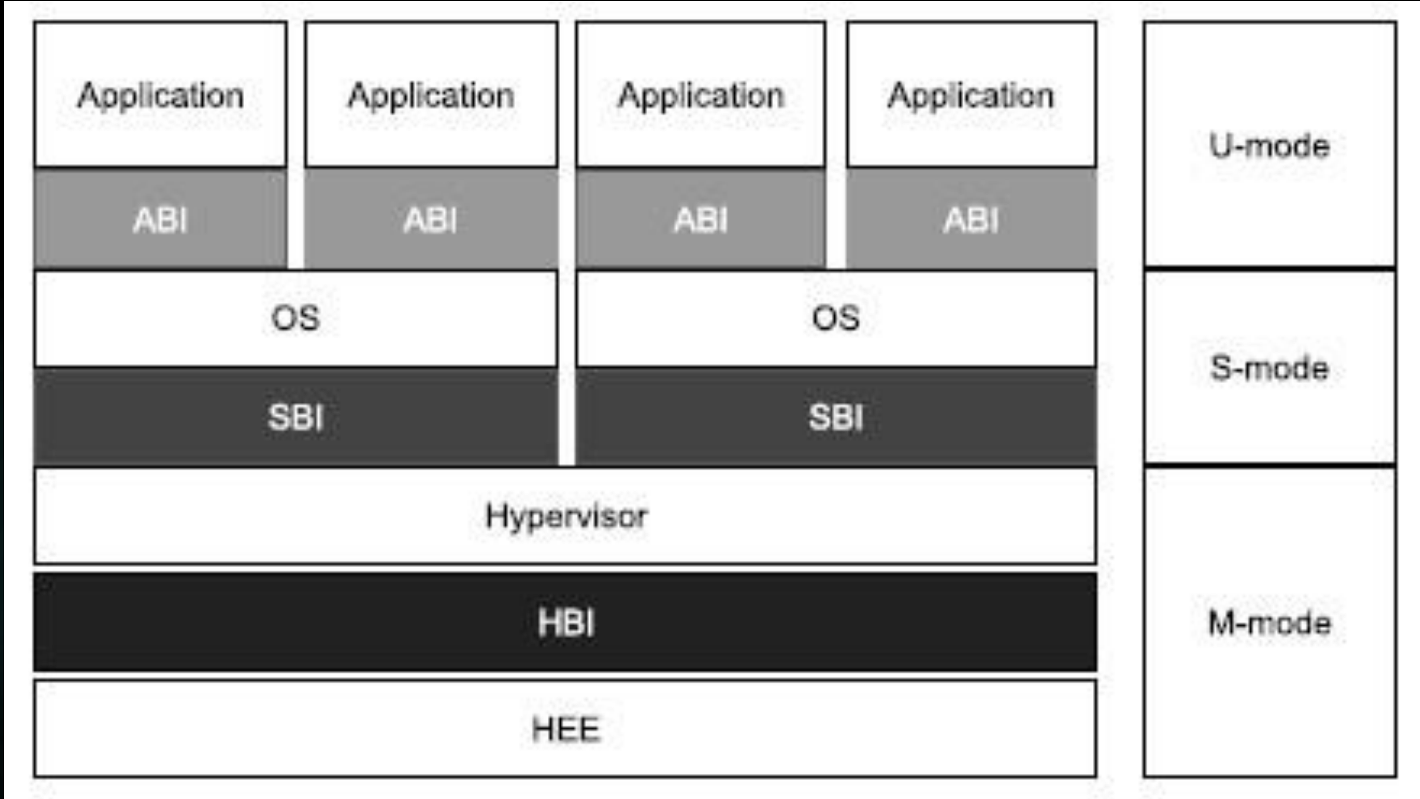
*« A **Trusted Execution Environment (TEE)** is a tamper-resistant processing environment that runs on a separation kernel. It guarantees the authenticity of the executed code, the integrity of the runtime states (e.g. CPU registers, memory and sensitive I/O), and the confidentiality of its code, data and runtime states stored on a persistent memory. In addition, it shall be able to provide remote attestation that proves its trustworthiness for third-parties. The content of TEE is no static; it can be securely updated. The TEE resists against all software attacks as well as the physical attacks performed on the main memory of the system. Attacks performed by exploiting backdoor security flaws are not possible. »*


Higher Privilege
Untrusted
Enclave 1
Enclave 2
User
(U-mode)
App
...
App
Enclave App 1
(Eapp)
...
Eapp 2
Supervisor
(S-mode)
Operating System
(OS)
Runtime (RT) 1
RT 2
Machine
(M-mode)
Security Monitor (SM)
Trusted
Hardware
RISC-V Cores
Optional H/W Features
Root of Trust

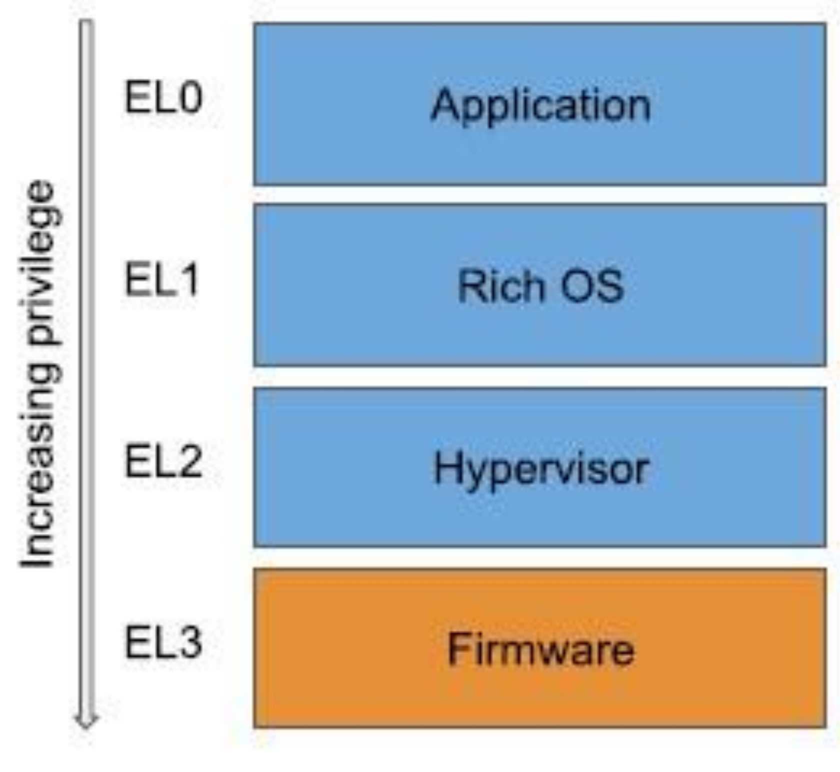
## Separation Kernel Architecture

- **Hypervisor Execution Enivironment (HEE):** its core runs in the highest security level (M-mode). It supports multiple OS and manages the TEE settings as well as the I/O tile.
- **Supervisor Execution Enivironment (SEE):** it is related to a specific OS and can be intended as a BIOS-like I/O system. It handles events such as interrupts, supervisor calls, traps and contain trusted applications.
- **Application Execution Enivironment (AEE):** it is the lowest level of security, in which trusted and untrusted applications run.





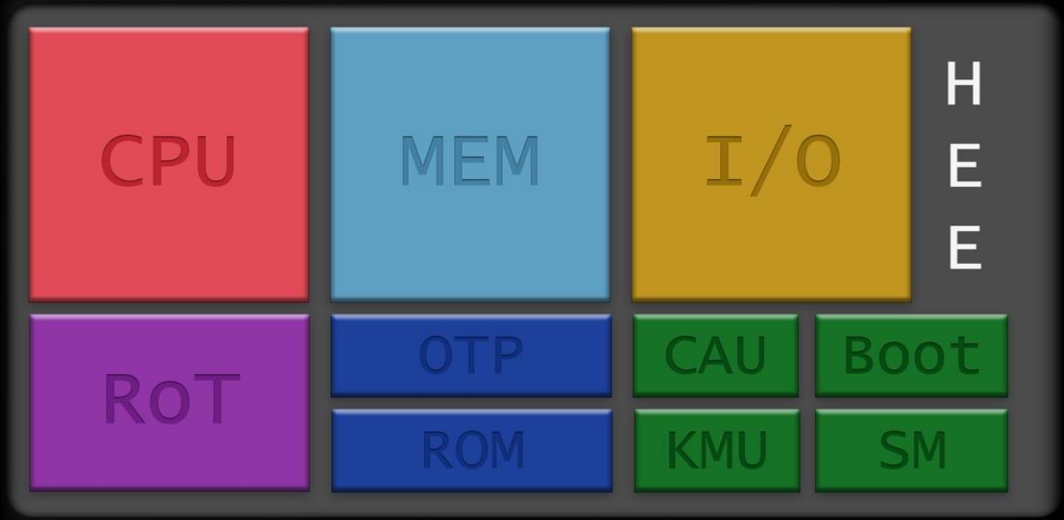
(a) RISC-V.



(b) Armv8-A.

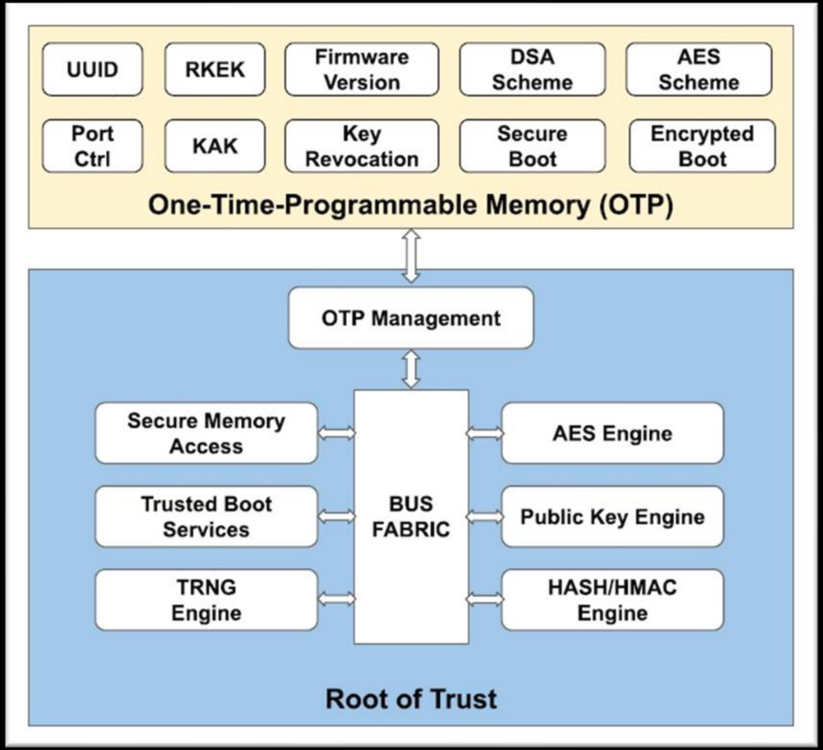
## TEE Building Blocks

- The building blocks of a TEE are:
  - Root of Trust (RoT)
  - Secure System Boot
  - Secure Scheduling
  - Inter-environment Communication
  - Secure Storage and Security Monitor (SM)
  - Trusted I/O Path



# The Root of Trust (RoT)

- The **Root of Trust (RoT)** is the foundation on which all secure operations of computing system depend. It must be secure by design since it will most likely be the target of the attacks.
- The building blocks of the RoT are:
  - Symmetric Cryptographic Engine
  - Asymmetric Cryptographic Engine
  - HASH Engine
  - Secure Memory
  - OTP Management Module
  - True Random Number Generator
  - Trusted Boot Services

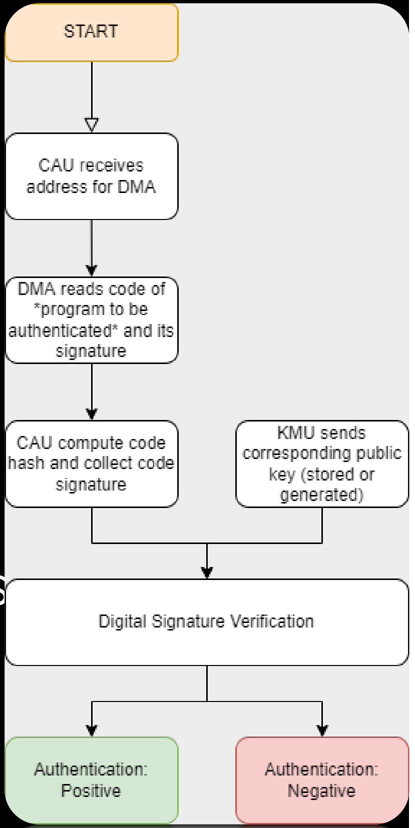


## Digital Signature Algorithm (DSA)

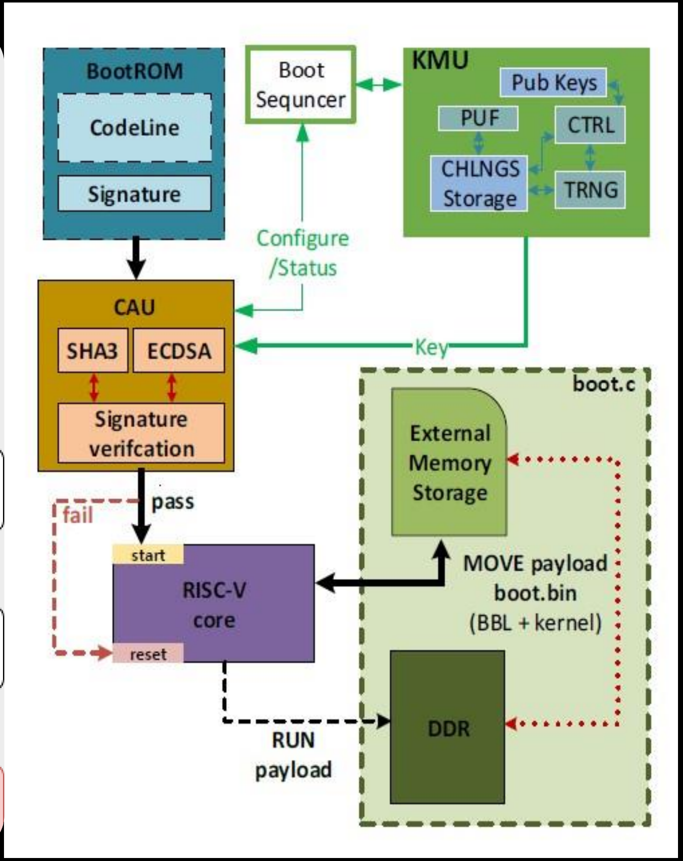
- It is a set of algorithms used to verify the digital signature of the code stored in the system and to be executed.
  - Key Generation Algorithm: provides a pair of a public and a private key randomly selected through a Physically Unclonable Function (PUF).
  - Signing Algorithm: receives as input the message and a private key and produces as output the message’s signature.
  - Signature Veryfing Algorithm: receives as input the message and its signature and a public key. It produces a binary output which either validate the signature or not.

# KMU, CAU and Boot Sequencer

- The **Code Authentication Unit (CAU)**: implements the digital signature algorithm in order to provide code authentication.
- The **Key Management Unit (KMU)**: generates and distribute symmetric and asymmetric keys stored in OTP Memory or generated through the **Physical Unclonable Function (PUF)** module.
- The **Boot Sequencer** is a FSM which implements the entire flow of the boot sequence.

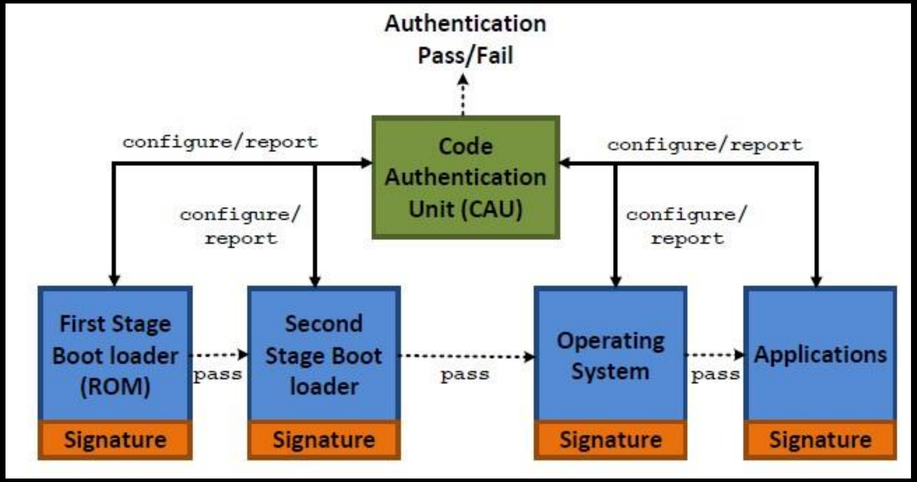


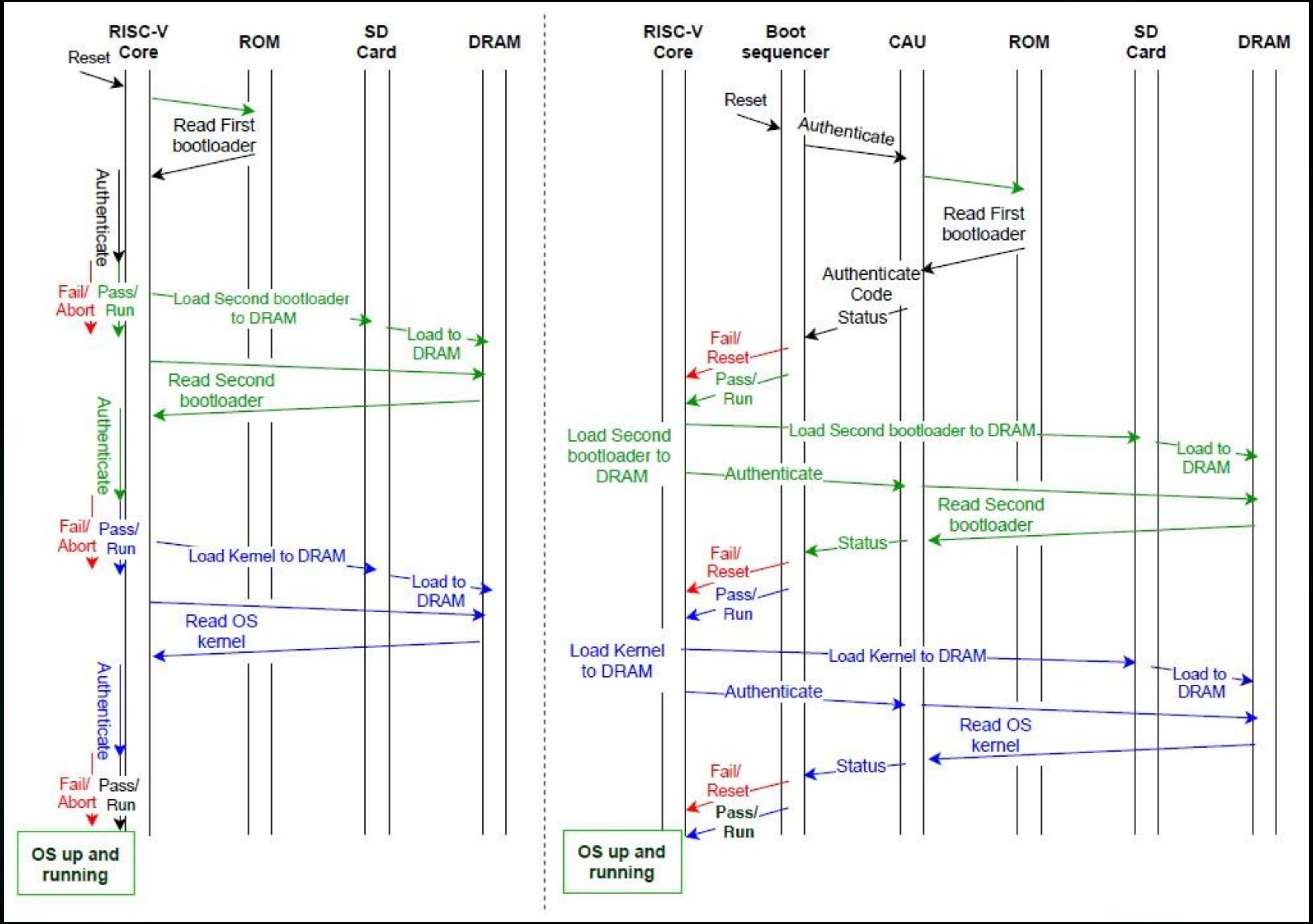
START
CAU receives address for DMA
DMA reads code of *program to be authenticated* and its signature
CAU compute code hash and collect code signature
KMU sends corresponding public key (stored or generated)
Digital Signature Verification
Authentication: Positive
Authentication: Negative



# Trusted System Boot Sequence

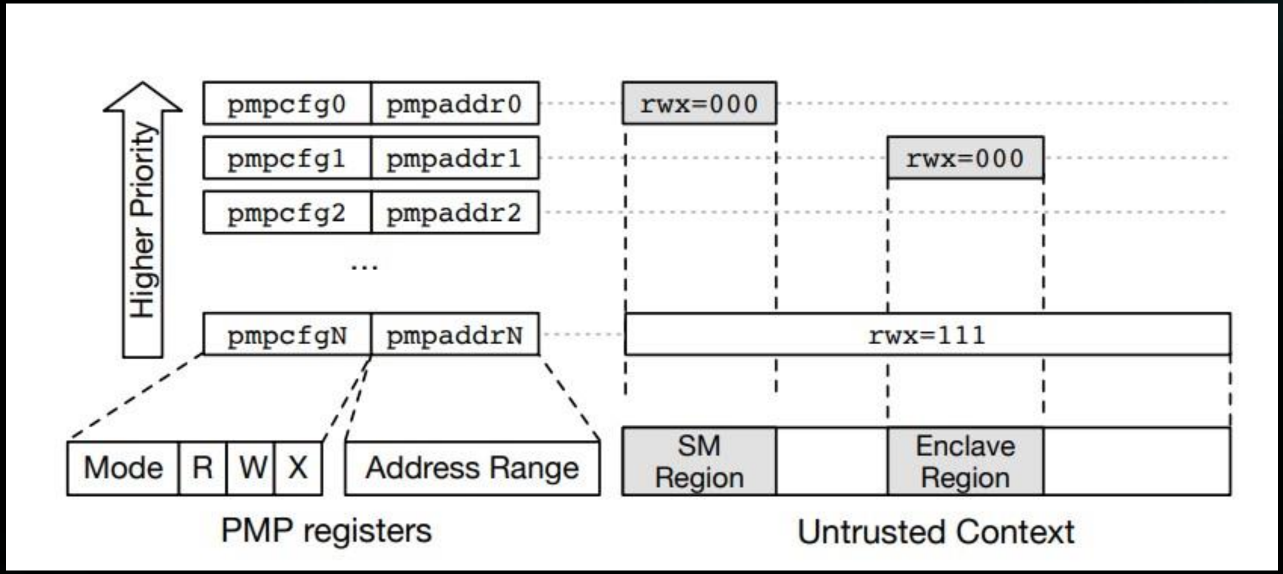
- Through the CAU every stage of the boot sequence is first authenticated and only then executed. If any step fails the entire Boot Sequence is stop or reset.
- The chain starts with an immutable first stage bootloader stored in the ROM/OTP memory. The second stage contains the Berkley Bootloader (BBL) which initialize peripherals, set-up page table, virtual memory and load and authenticate the kernel. If this operation is again successful the kernel code is executed and the OS is up and running.




RISC-V Core
ROM
SD Card
DRAM
Reset
Read First bootloader
Authenticate
Fail/ Abort
Pass/ Run
Load Second bootloader to DRAM
Load to DRAM
Read Second bootloader
Authenticate
Fail/ Abort
Pass/ Run
Load Kernel to DRAM
Load to DRAM
Read OS kernel
Authenticate
Fail/ Abort
Pass/ Run
OS up and running
RISC-V Core
Boot sequencer
CAU
ROM
SD Card
DRAM
Reset
Authenticate
Read First bootloader
Authenticate Code Status
Fail/ Reset
Pass/ Run
Load Second bootloader to DRAM
Load Second bootloader to DRAM
Load to DRAM
Authenticate
Read Second bootloader
Status
Fail/ Reset
Pass/ Run
Load Kernel to DRAM
Load Kernel to DRAM
Load to DRAM
Authenticate
Read OS kernel
Status
Fail/ Reset
Pass/ Run
OS up and running

## Physical Memory Protection (PMP)

- It is a memory protection technology which allows the firmware to limit accesses to memory regions (Read R, Write W, Execute X) and lock it to unprivileged running cores (S and U mode) and if necessary, even the M-mode core itself.
- A locked region is called **Enclave** and they are set through the Control Status Registers (CSR) (up to 16 per core).
- Each access is controlled through a PMPChecker which is an in-core module which checks every access to memory regions.
- Enclave are handled by the **Security Monitor (SM)** which refer to the HEE every trap or fault and at the end of the life-cycle of the enclave, clear its memory region.



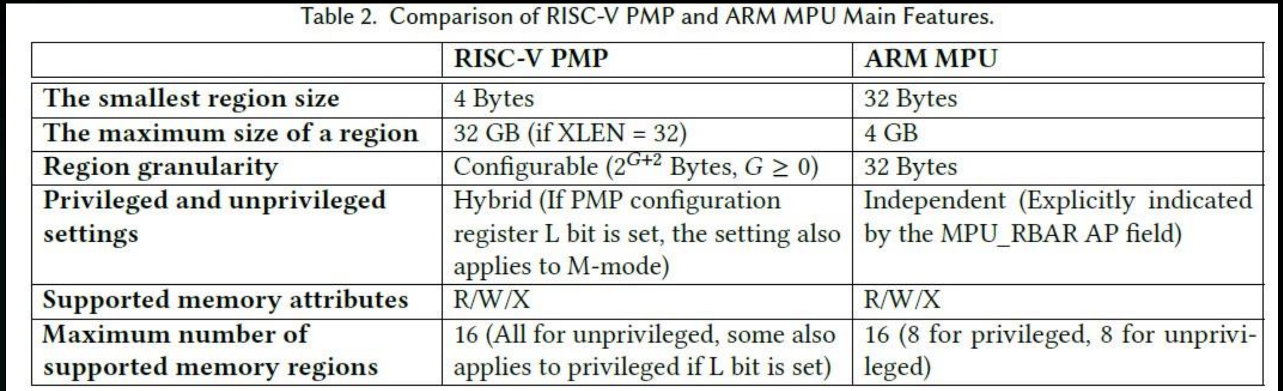


Table 2. Comparison of RISC-V PMP and ARM MPU Main Features.

| | RISC-V PMP | ARM MPU |
|---|---|---|
| **The smallest region size** | 4 Bytes | 32 Bytes |
| **The maximum size of a region** | 32 GB (if XLEN = 32) | 4 GB |
| **Region granularity** | Configurable ($2^{G+2}$ Bytes, $G \geq 0$) | 32 Bytes |
| **Privileged and unprivileged settings** | Hybrid (If PMP configuration register L bit is set, the setting also applies to M-mode) | Independent (Explicitly indicated by the MPU_RBAR AP field) |
| **Supported memory attributes** | R/W/X | R/W/X |
| **Maximum number of supported memory regions** | 16 (All for unprivileged, some also applies to privileged if L bit is set) | 16 (8 for privileged, 8 for unprivileged) |

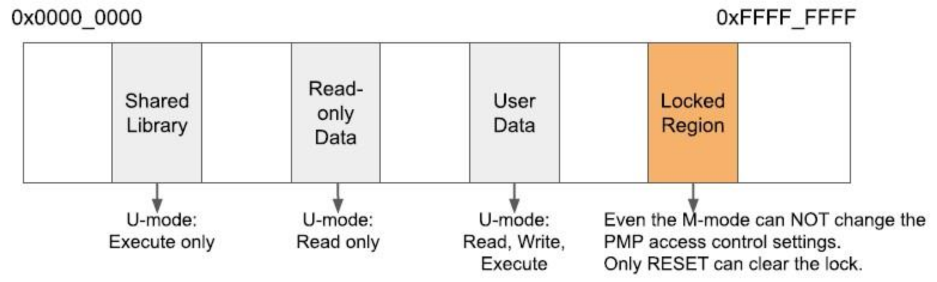
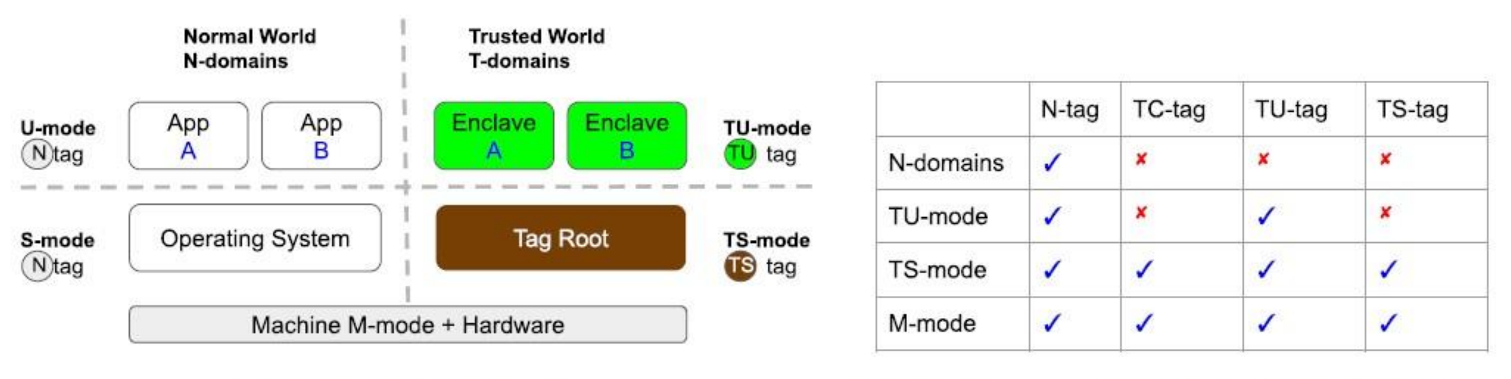

0x0000_0000
0xFFFF_FFFF
Shared Library
Read-only Data
User Data
Locked Region
U-mode: Execute only
U-mode: Read only
U-mode: Read, Write, Execute
Even the M-mode can NOT change the PMP access control settings. Only RESET can clear the lock.

Fig. 4. Demonstration of RISC-V physical memory protection.

# Tagged Memory

- Differently from the PMP, it directly modify the words in the memory by adding a predefined number of bits. The word extension is also implemented in cache levels to keep system coherence.
- At every memory access, the memory address will be check to prevent any capability violation.



| | N-tag | TC-tag | TU-tag | TS-tag |
|---|---|---|---|---|
| N-domains | ✓ | ✘ | ✘ | ✘ |
| TU-mode | ✓ | ✘ | ✓ | ✘ |
| TS-mode | ✓ | ✓ | ✓ | ✓ |
| M-mode | ✓ | ✓ | ✓ | ✓ |

(a) Four security domains. (b) Tag update permission in each security domain.

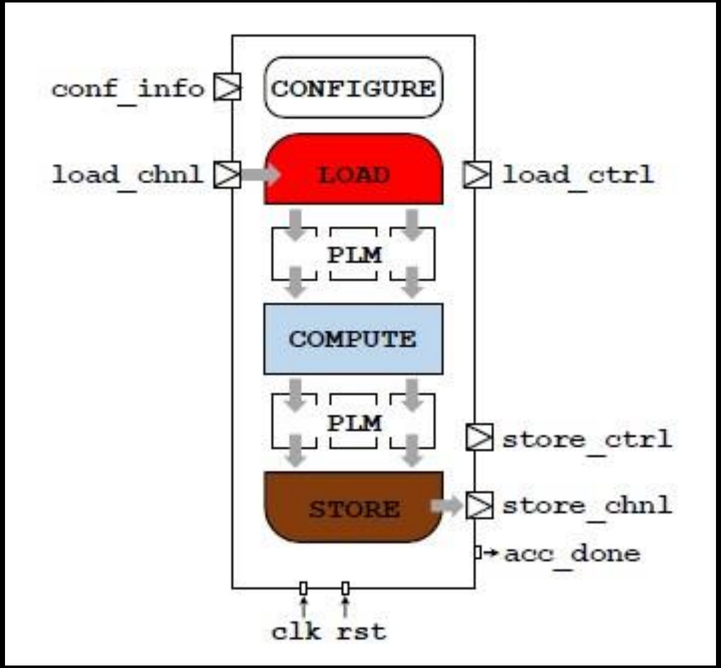
Fig. 8. TIMBER-V Fine-grained Enclaves based on Tag-Isolated Memory. This figure is adopted from paper [71].

## Encryption in TEE

- It is fundamental in most of the security technologies used in a TEE (for example Digital Signature Algorithm, HASH function, memory encruption).
- To keep the security/performance trade-off convenient encryption functions must be included in the Instruction Set Architecture (ISA), as well as the implementation through hardware-assisted solutions.

# Hardware Accelerators Integration

- It is mandatory to prevent the attacker obtain sensible data by exploiting the Direct Memory Access (DMA) and the P2P interconnection system implemented in each hardware accelerator tile.
- To obtain this, the core which is calling the hardware accelerator must send a tag as well as setting a running mode toghether with input data and other configuration settings.
- Each core can impose to the HwA a security level which is equal or lower than its one. In this way, memory privileges escalation is prevented.



## Oblivious RAM (ORAM)

- It is implemented to transform algorithms in such a way that the resulting one keep the same input/output behavior but the distribution of memory access pattern is independent of the memory access pattern of the original algorithm.
- It prevents the attacker to obtain sensible informations by simply observing the pattern of memory accesses.

## Side-channel attacks prevention

- Hardware solutions: redesing of cache and processor architecture to improve cross-processor information flow tracking.
- Software solutions: clear all core states such as private caches, translation backup buffers, branch prediction units through a refresh mechanism.
- To prevent Timing attacks it is useful to implement refresh and clear operation to ISA.
- Power Analysis attacks can be prevented through Dynamic Frequency Scaling (DFS).

Grazie per l’attenzione!