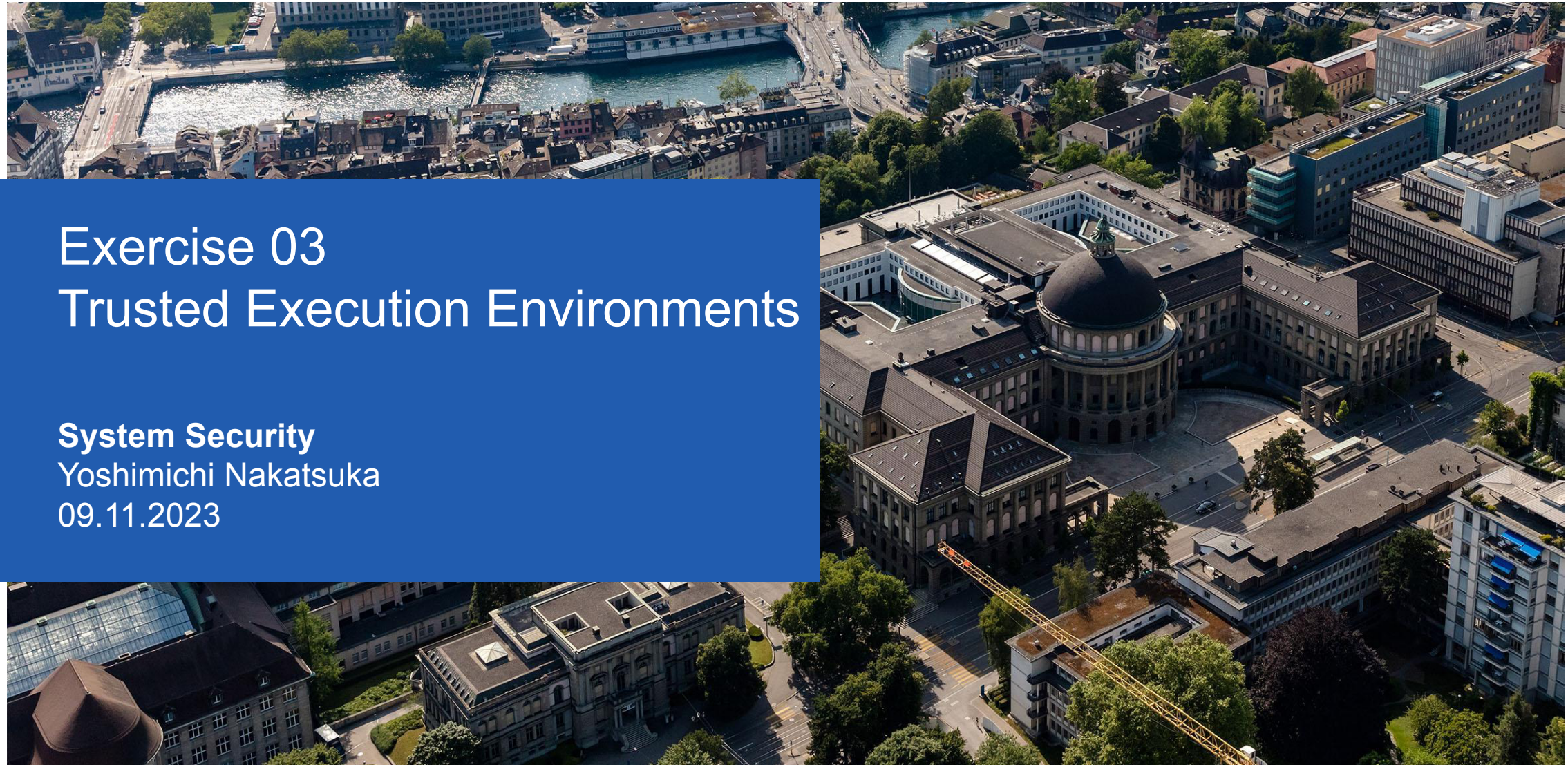
ETH zürich



Exercise 03
Trusted Execution Environments

System Security

Yoshimichi Nakatsuka

09.11.2023

Adapted from: 2022 TEE slides (credits: Friederike Groschupp)

Agenda

1. TEE fundamentals
2. Side-/Controlled-Channel Attack on SGX
3. Recent TEE developments
4. Remarks about Exercise 4

TEE Fundamentals

What is a Trusted Execution Environment (TEE)?

SoK: Hardware-supported Trusted Execution Environments

Moritz Schneider, ETH Zurich
Ramya Jayaram Masti, Intel Cooperation
Shweta Shinde, ETH Zurich
Srdjan Capkun, ETH Zurich
Ronald Perez, Intel Cooperation

“Oddly, despite the abundant research on TEEs and the growing number of commercially available TEE solutions, there is no single, widely-accepted definition for a TEE.”

What is the TCB of an application?

- All components that need to be trusted and bug-free for the secure and correct execution of the application.
- The bigger the TCB, the harder to verify that it does what it is supposed to do and that it is bug-free.

What is the rationale for using Trusted Execution Environments?

- Reduce the TCB of security-sensitive applications
- Often: remove OS and hypervisor from the TCB
- Applications: Cloud computing, DRM, secure storage on smartphones

What are the three TEE primitives? Briefly explain their purpose.

- Isolation: Protect the Integrity and confidentiality of an applications execution and data
- Bootstrapping Trust: Assert that the correct software is running on correct (TEE) hardware before interacting with it or sending secrets to it.
  - Attestation: Measure an enclave at boot time, sign an attestation report with a manufacturer embedded, secret private key that contains the measurement, freshness/nonce/”message” from enclave, system/TCB version, and the public key certificate
  - Secure Boot: Chain of trust from the first boot stages to the trusted components
- Sealing: Enable TEEs to store data confidentiality- and integrity-protected over several runs and boots.

What are the three TEE primitives? Briefly explain their purpose.

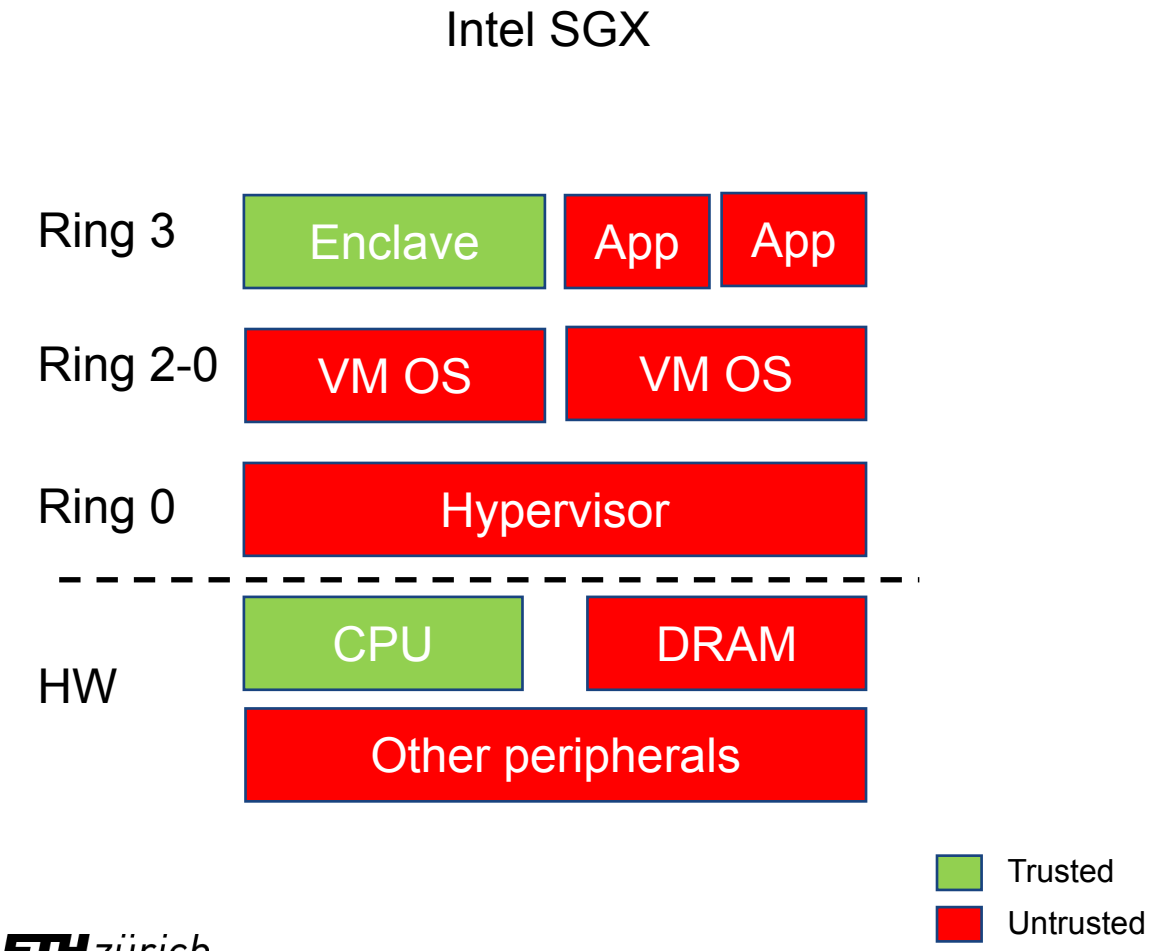
- Note: This answer depends on who you ask, e.g., from the SoK:

  “While existing TEEs often vary in terms of their exact security goals, most of them aim to provide (a subset of) four high level security protections

  i. verifiable launch of the execution environment for the sensitive code and data so that a remote entity can ensure that it was set up correctly,
  ii. run-time isolation to protect the confidentiality and integrity of sensitive code and data
  iii. trusted IO to enable secure access to peripherals and accelerators, and finally,
  iv. secure storage for TEE data that must be stored persistently and made available only to authorized entities at a later point in time“

What are the trust assumptions of Intel SGX, AMD SEV and ARM TrustZone, respectively?

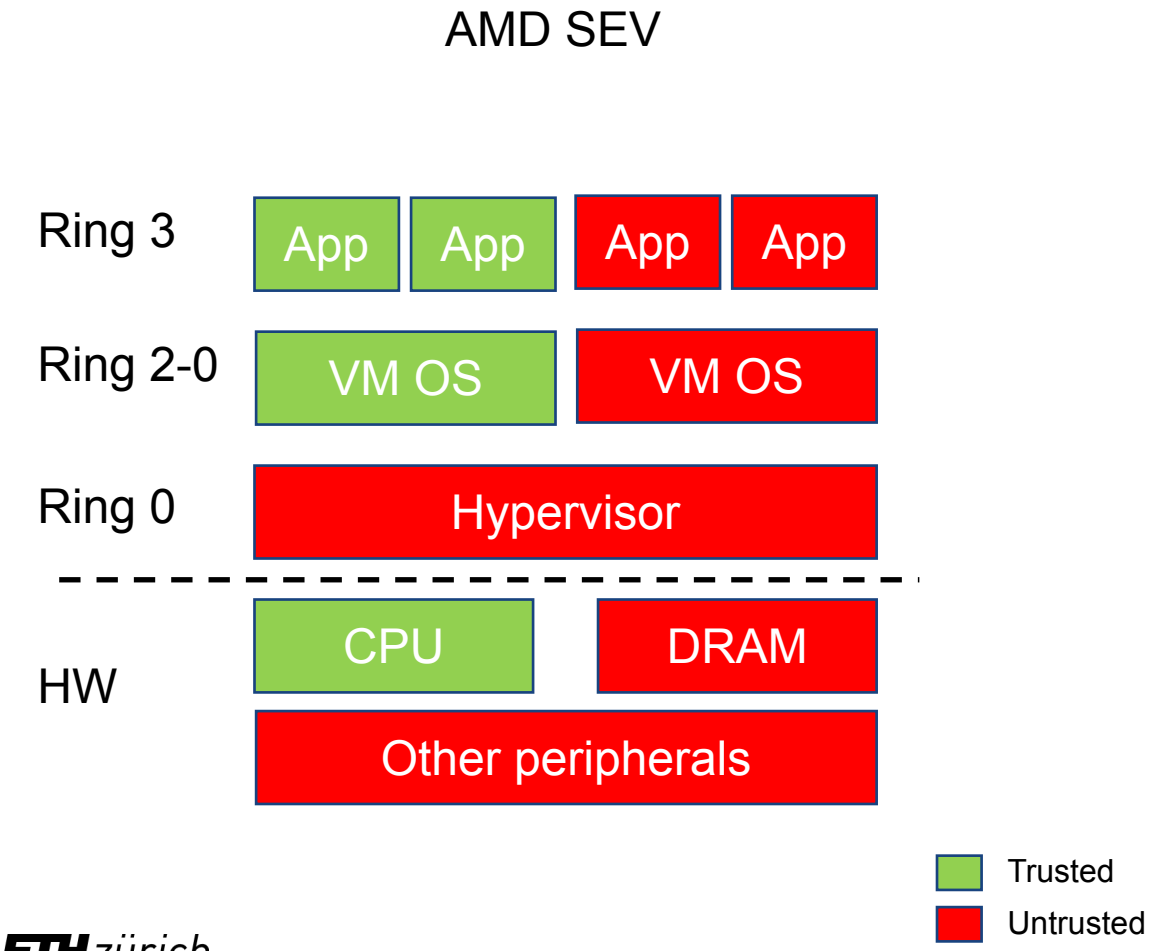
How does each of them enforce runtime isolation?



- Hardware tracks if and which enclave is currently executing
- Enclave memory only accessible if enclave is executing
- Memory outside CPU die is encrypted
- Strong physical attacker considered

What are the trust assumptions of Intel SGX, AMD SEV and ARM TrustZone, respectively?

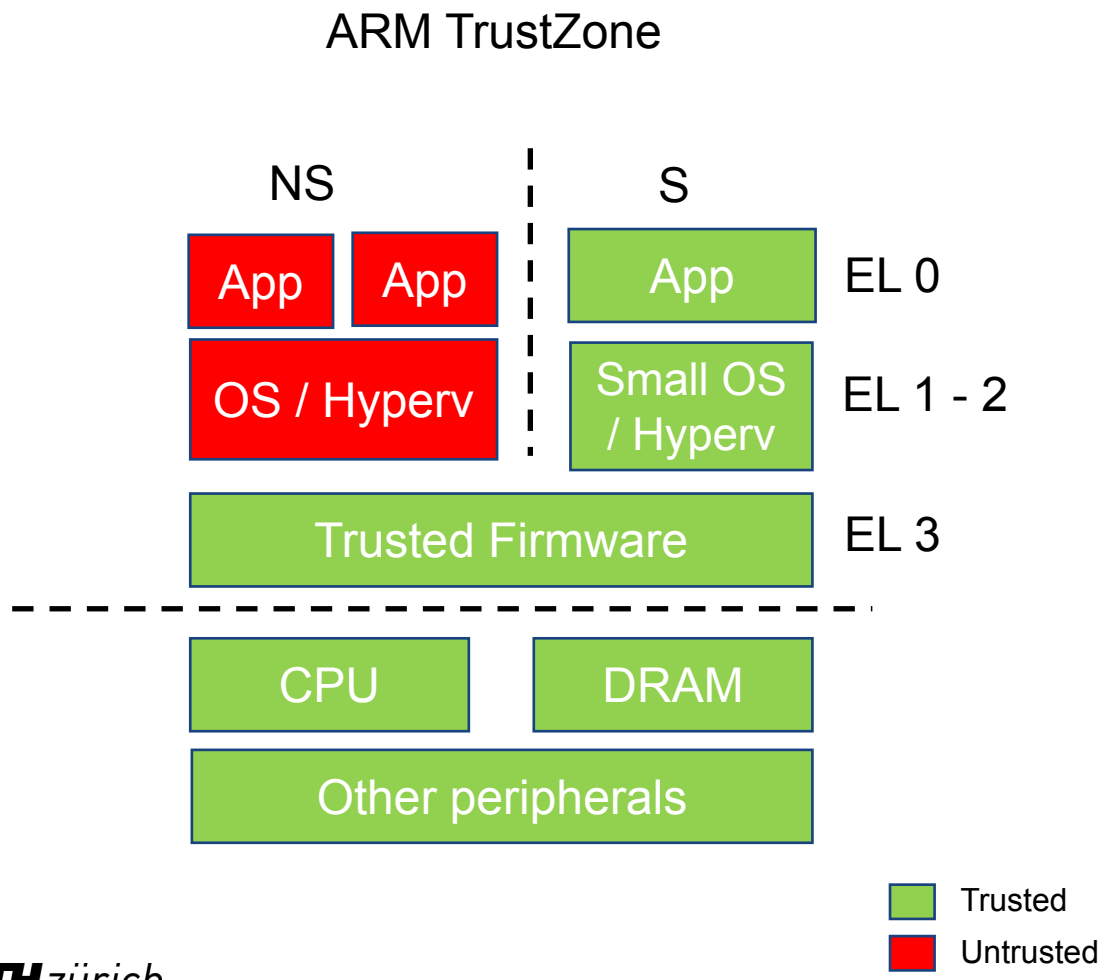
How does each of them enforce runtime isolation?



- Isolation at VM-level
- Every VM‘s memory is encrypted with its own key
- Encryption support provided by co-processor
- Tolerated physical attacker strength depends on SEV version

What are the trust assumptions of Intel SGX, AMD SEV and ARM TrustZone, respectively?

How does each of them enforce runtime isolation?



- Isolation of EL2 to EL1 into „two worlds“, one secure one non-secure/normal
- Based on NS bit, which is propagated on bus
- Context switches managed by software in EL3
- Memory Isolation etc. must be enforced on subordinate side
- No protection against physical attacker per se

What are the advantages and disadvantages of SGX, SEV, and TrustZone?

- SGX
  - Tolerates strong physical attacker
  - Only CPU and microcode in TCB (plus Intel for attestation)
  - Memory and interrupt management performed by OS
  - Applications need to be adapted and are restricted in what they can do, no secure I/O
  - …
- SEV
  - Applications do not need to be (significantly) adapted
  - Memory management performed by hypervisor
  - (Resilience against physical attacker varies by SEV version)
  - …

What are the advantages and disadvantages of SGX, SEV, and TrustZone?

- TrustZone
  - No interrupt / memory management by untrusted component
  - Platform infrastructure partially supports secure state
  - Possibility for availability guarantees
  - Physical attacker not considered, open to manufacturer implementation
  - No general attestation scheme
  - Only one „protected environment“
  - Rather large software TCB
  - Secure state often locked for use by manufacturer

Page-based Controlled Channel Attack on SGX

Which information about the execution of an SGX enclave does the paper use for the side-channel attack?

- Page trace built from intentionally triggered page faults

Which steps does the malicious OS need to take to obtain this information?

- Remove execute/read/write permissions from all or selected pages
- When enclave tries to access this page, a page fault is triggered which is routed to the OS
- Page fault contains address of page, note it
- Grant permission, continue
- On next page fault, withdraw access rights again

How can the OS use this information to infer what is being executed inside the enclave?

- Precondition: Page accesses depend on input, e.g. sensitive data or key material
- Observe the page trace
- Reconstruct possible input data from page trace, e.g. by comparing it to recorded traces

Let's look at a minimal example!

Consider an SGX enclave E. E accepts an AES-encrypted message `cipher` together with a counter value `ctr` through an ECALL, decrypts the message using a secret `key` stored within the enclave, and then processes the resulting plaintext further. The ECALL can be called arbitrarily often. For the purpose of this question, we assume that `key`, `cipher`, and `ctr` are all 8 byte (64 bit) long. The first steps of the decryption process can be abstracted in the following way:

```
decrypt(byte[8] ctr, byte[8] cipher, byte[8] key){
    byte[8] temp = {0};
    //first step of the first round of AES in counter mode
    for (int i = 0; i < 8; i++) {
        // '^' is XOR
        temp[i] = sbox[ctr[i] ^ key[i]];
    }
    //continue with the rest of the AES steps and rounds...
}
```

`sbox` is a byte array of length 256 that is stored statically in enclave memory. The array is distributed among two memory pages: The first half (indices from 0x00 to 0x7F) is on page P_0, the second half (indices from 0x80 to 0xFF) is on page P_1.

Let's look at a minimal example!

Consider an SGX enclave E. E accepts an AES-encrypted message `cipher` together with a counter value `ctr` through an ECALL, decrypts the message using a secret `key` stored within the enclave, and then processes the resulting plaintext further. The ECALL can be called arbitrarily often. For the purpose of this question, we assume that `key`, `cipher`, and `ctr` are all 8 byte (64 bit) long. The first steps of the decryption process can be abstracted in the following way:

```
decrypt(byte[8] ctr, byte[8] cipher, byte[8] key){
    byte[8] temp = {0};
    //first step of the first round of AES in counter mode
    for (int i = 0; i < 8; i++) {
        // '^' is XOR
        temp[i] = sbox[ctr[i] ^ key[i]];
    }
    //continue with the rest of the AES steps and rounds...
}
```

`sbox` is a byte array of length 256 that is stored statically in enclave memory. The array is distributed among two memory pages: The first half (indices from 0x00 to 0x7F) is on page P_0, the second half (indices from 0x80 to 0xFF) is on page P_1.

Assume the secret `key` is **0x94 b7 b9 3b 6a 45 4b 44**, the user-provided `ctr` is **0x7c 9b 05 05 9b 28 3e fe**, and `cipher` is **0x34 c6 bb 26 fa 61 56 e9**. What is the page trace for P_0 and P_1 produced by the first steps of the decryption?

1. Figure out with which index sbox is accessed for each byte combination of ctr and key:

   94 b7 b9 3b 6a 45 4b 44 ^ 7c 9b 05 05 9b 28 3e fe = E8 2C BC 3E F1 6D 75 BA

2. Figure out which page is accessed for which index values of sbox:

   0x00 – 0x7F: P_0

   0x80 – 0xFF: P_1

3. Combine information:

E8	2C	BC	3E	F1	6D	75	BA
P1	P0	P1	P0	P1	P0	P0	P1

Provide a key which does not have any byte values in common with the original key, but produces the same page trace under the assumption that ctr and cipher stay the same.

E8	2C	BC	3E	F1	6D	75	BA
P1	P0	P1	P0	P1	P0	P0	P1

Condition for each byte: first bit of ctr[i] ^ key[i] must be 0 or 1 to access page P0 or P1, respectively

Example:

0x95 b8 ba 3c 6b 46 4c 46

Which conclusions can an attacker draw about the key from the observed page trace? How many brute force attempts does the attacker need to do in the worst case given the side channel information they obtained?

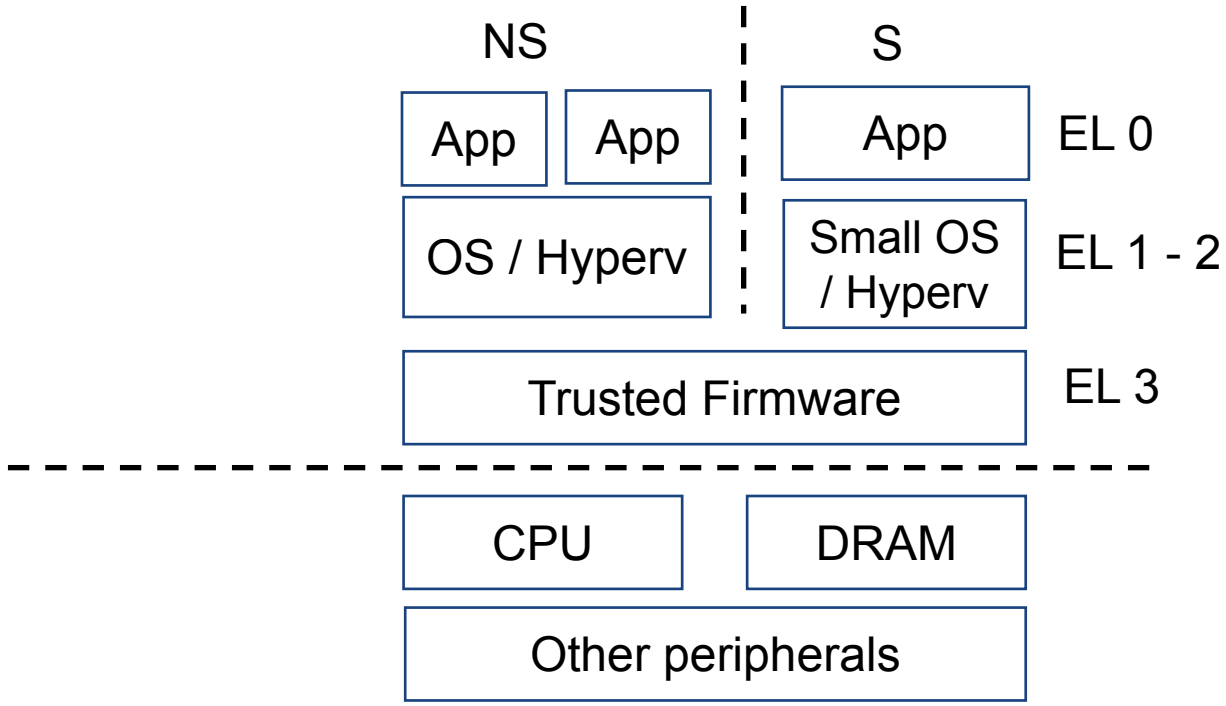
- From page trace, attacker can deduce for each byte the first bit
- This leaves $(2^7)^8 = 2^{56}$ possible keys instead of the initial 2^{64}

Bonus question I: Is there a way for the attacker to find out more information about the key?

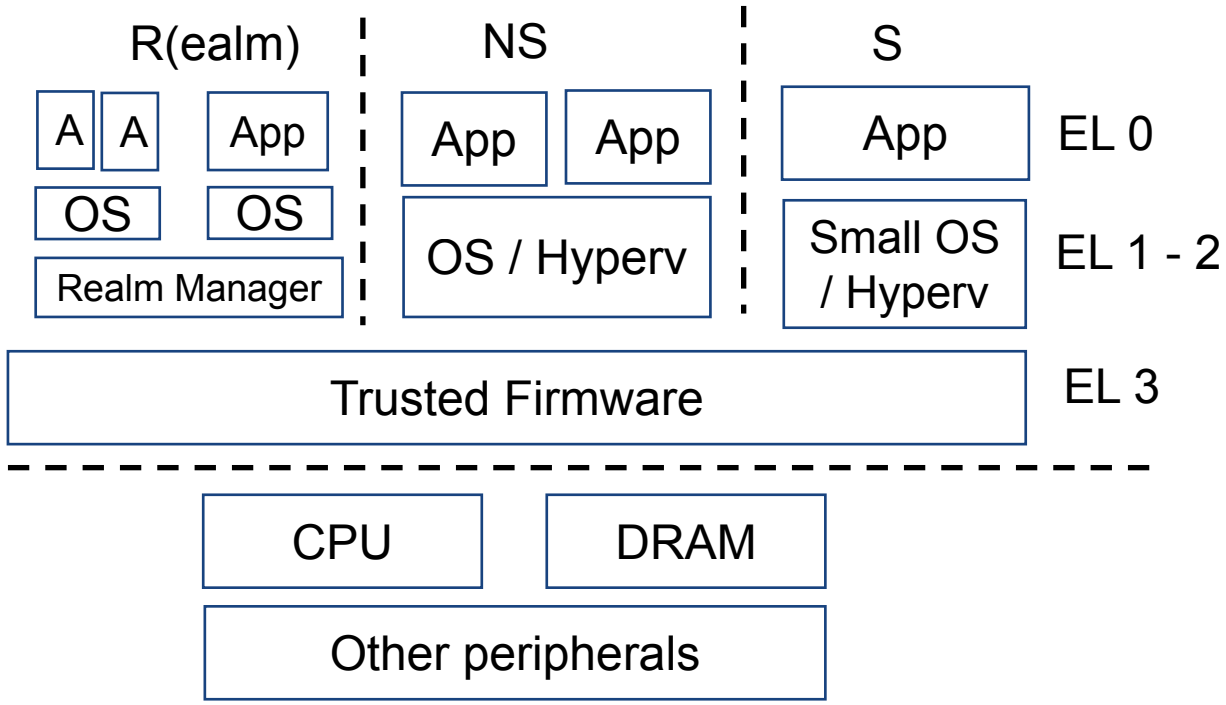
Bonus question II: We assume that we can observe one page fault per access. Which condition needs to hold for this to be actually the case? Think about how the attacker triggers the page faults.

Recent TEE developments

ARM Realm Management Extensions - Design

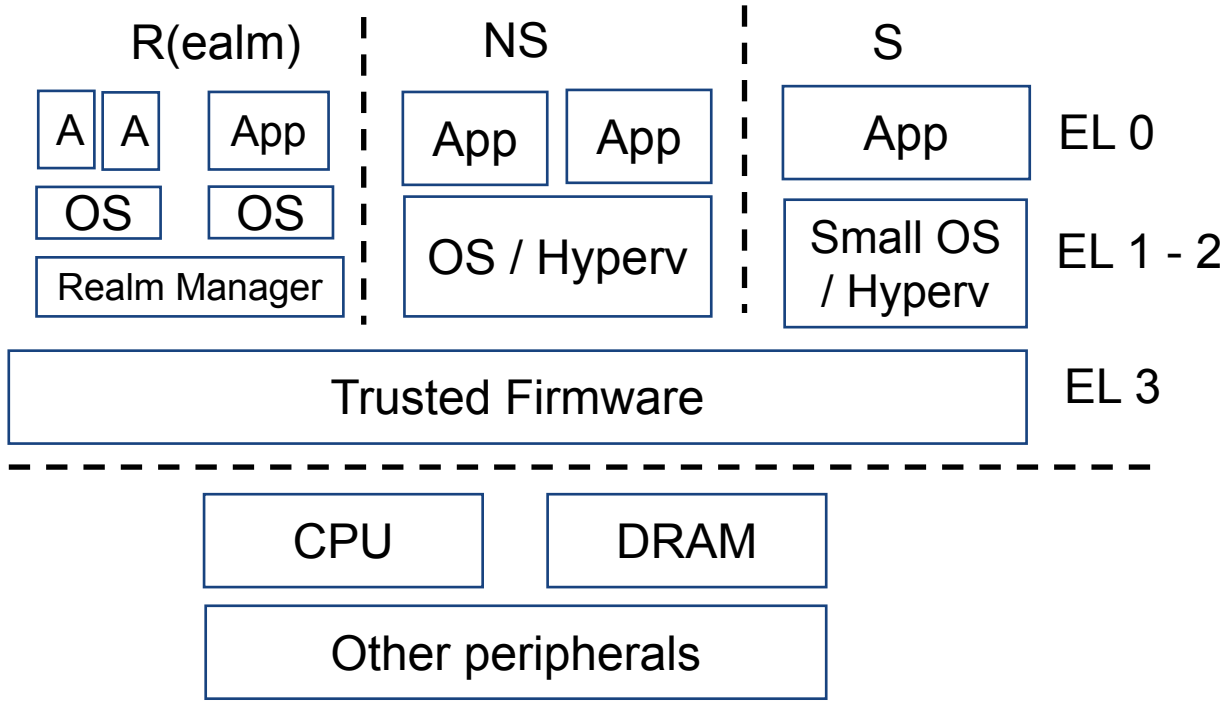


ARM Realm Management Extensions

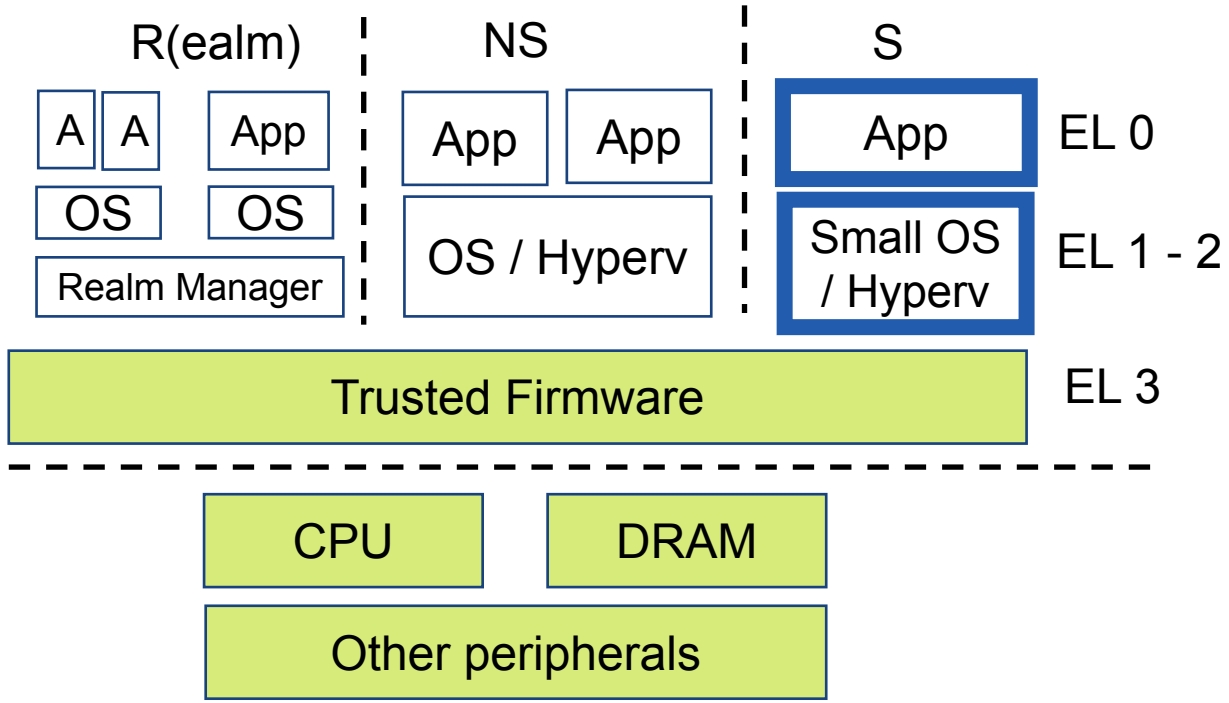


- Two new security states
- Isolation between states enforced by Granule Protection Table (managed by EL3)
- Isolation between realms enforced by stage 2 translation tables (controlled by R-EL2)

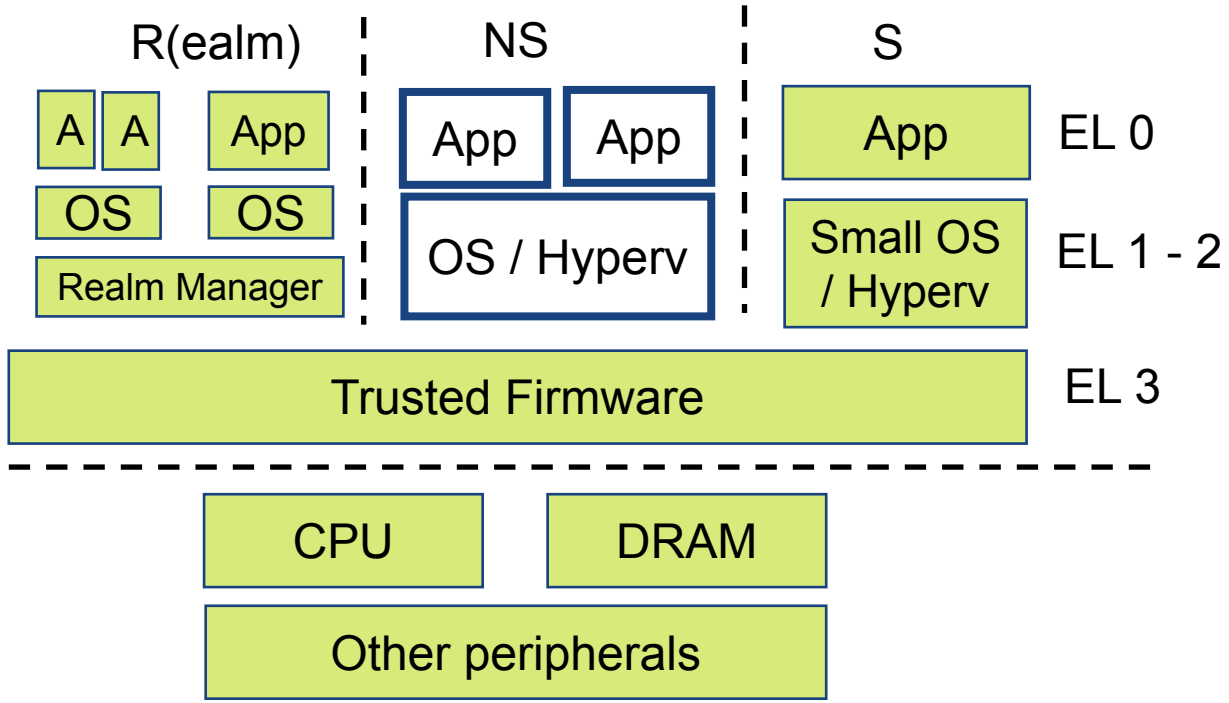
ARM Realm Management Extensions –TCB



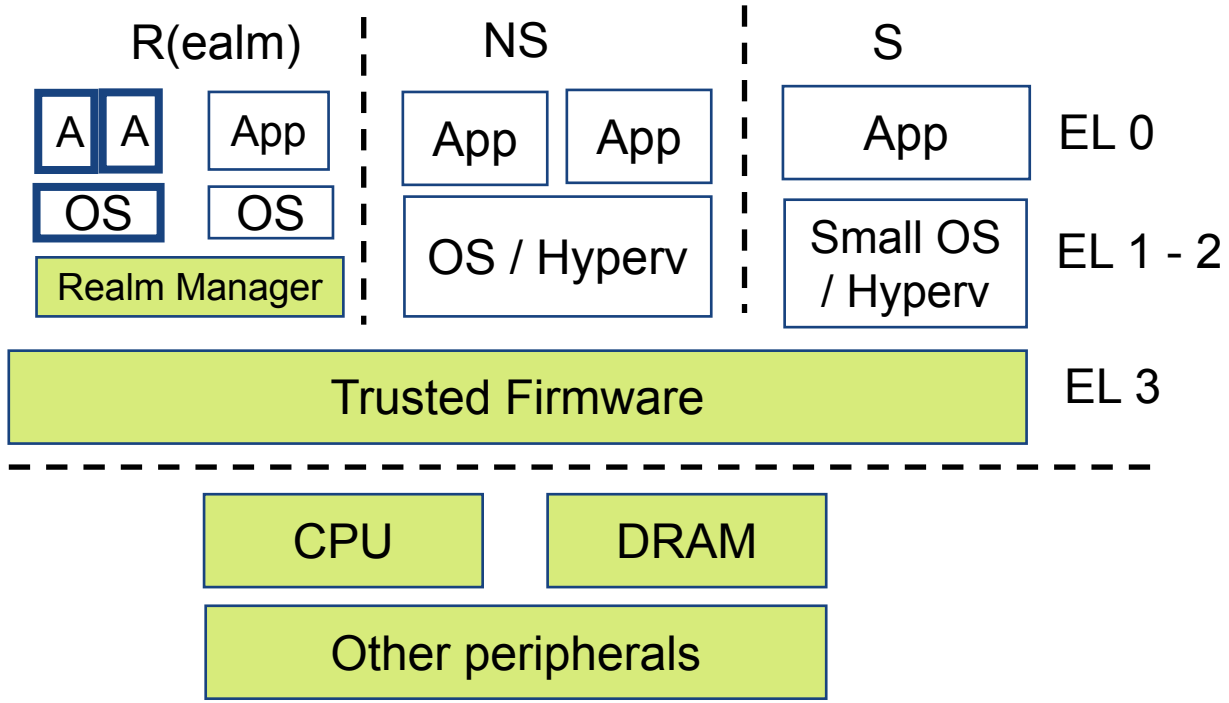
ARM Realm Management Extensions –TCB



ARM Realm Management Extensions –TCB



ARM Realm Management Extensions –TCB



Which component is in charge of assigning resources to realms?

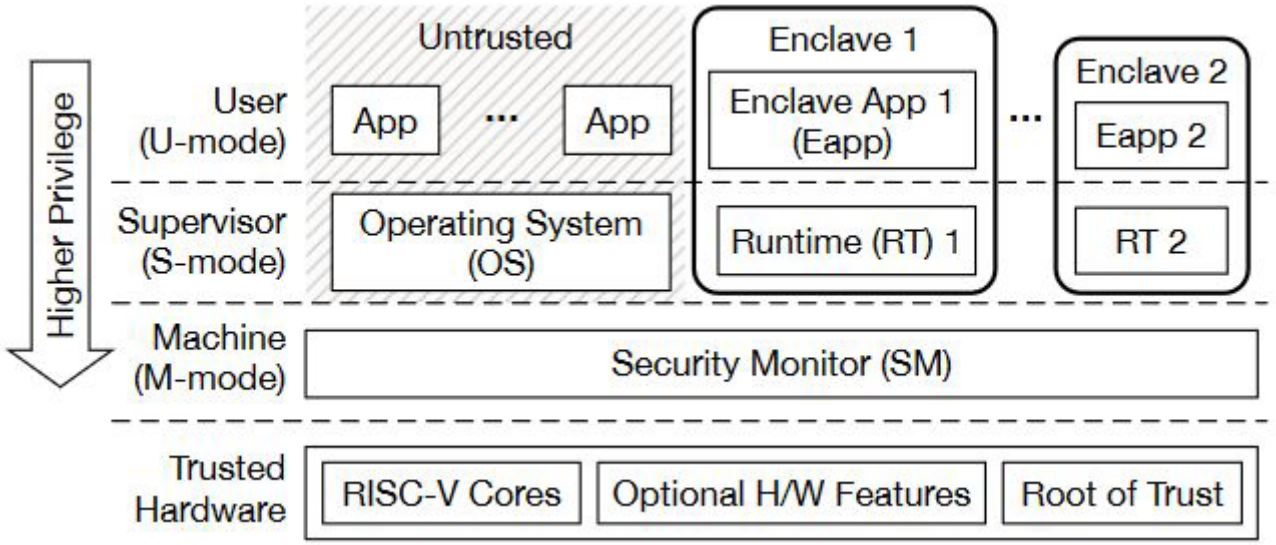
- Not strictly defined for Realm Extensions, depends on software implementation
- ARM plans to provide reference implementation where management is performed by non-secure world to keep realm TCB small

How does realm compare to TrustZone?

- New security state that allows deployment of „enclaves“ that are not provided/allowed by manufacturer
- New hardware block for memory block: Granule Protection Table (GPT)
- No availability guarantees for realm state, no support for realm state in other platform components

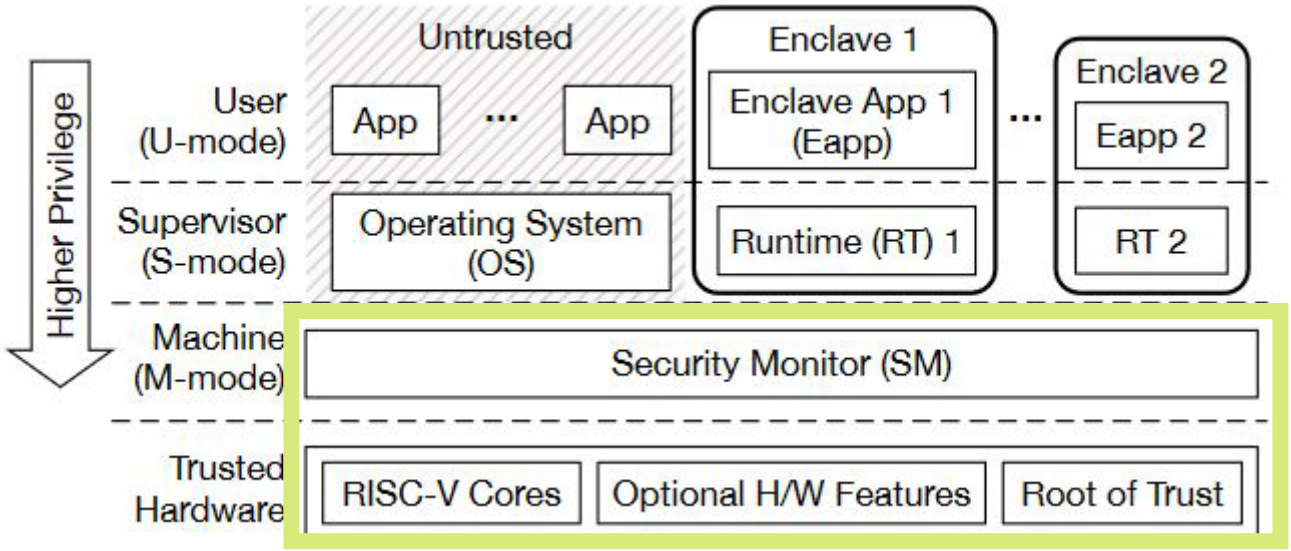
Do you see similarities to another TEE technology that you already know?

Keystone - Design

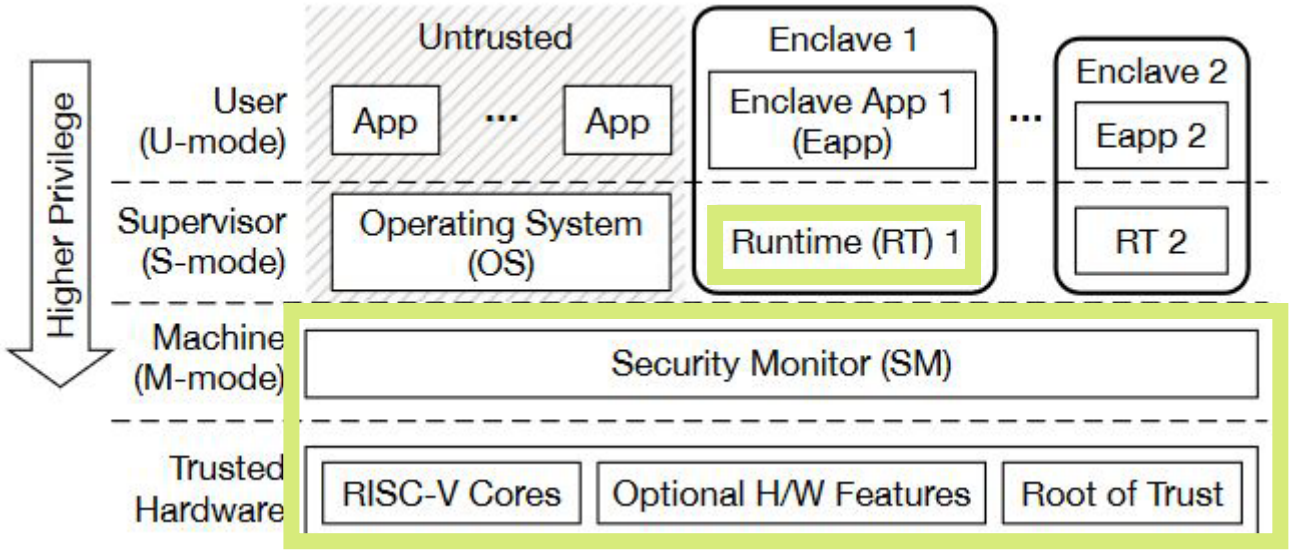


- Isolation of enclaves and host of through PMP entries, managed by security monitor

Keystone – TCB of untrusted OS



Keystone – TCB of eapp



Recall the side-channel attack on Intel SGX. Is such an attack still possible with Keystone? Why/why not?

- Memory management is performed by trusted runtime, which is part of the enclave
- -> OS cannot modify page table permissions, which is necessary for the attack

Do you see similarities to another TEE technology that you already know?

Exercise 4

Exercise 4 – General Information

- Intel SGX programming using SGX SDK
- No SGX HW required. We run the enclaves/apps in simulation mode.
- The exercise sheet provides a lot of guidance.
- Also check the SGX developer guide and the examples folder!
- Good resource: https://sgx101.gitbook.io/sgx101

Exercise 4 – Motivation

- Alice made a huge discovery! She found a way to add two integers.
- Bob claims that he has known a method for this problem for years!
- Alice demands proof, but Bob does not want to disclose his secret method.
- Each of them will implement an enclave, enclave A and enclave B, respectively.
- Enclave A will create challenges, i.e. a pair of Integers, and send them to enclave B.
- Enclave B will add the Integers and send the result back to enclave A.
- Enclave A verifies and outputs whether the result is correct.

Exercise 4 – Remarks

- Tasks:
  - Implement the challenge and response phase (App.cpp)
  - Design ECALLs with correct input and output arguments (Enclave.edl)
  - Implement all ECALL functions (Enclave.cpp)
- The relevant locations for this task are marked “EXERCISE: ” in the code-base.
- If you're having trouble with SGX data types, function signatures, or C in general: the SGX Github Repo, the Developer Reference, and the search engine of your choice are your friends.