TEE and its Key Management

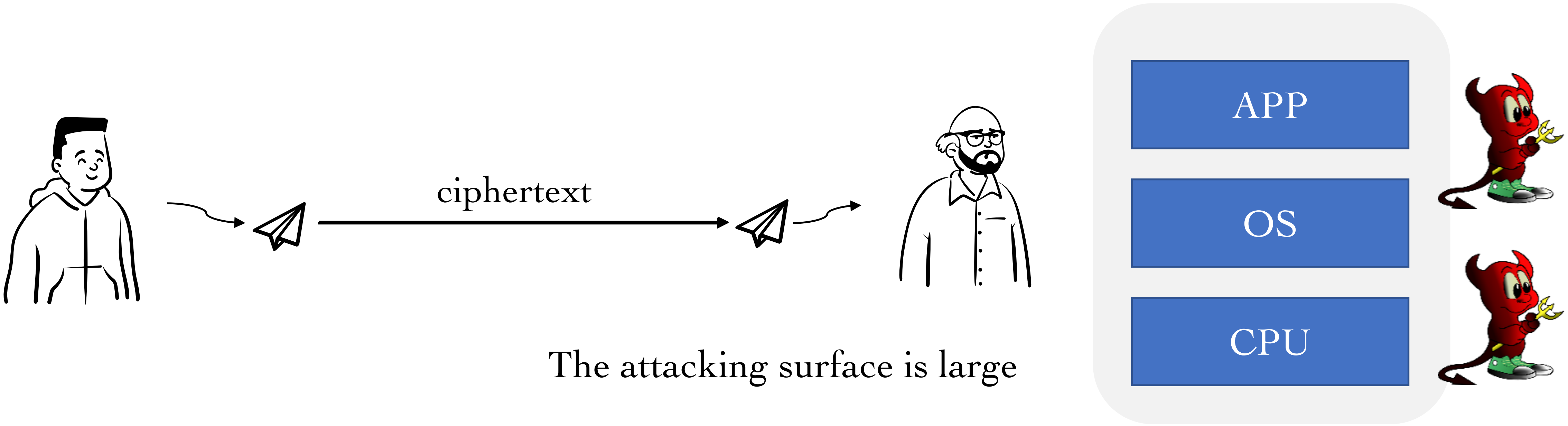
Rujia Li

2023/01/13

Institute for Advanced Study, Tsinghua University

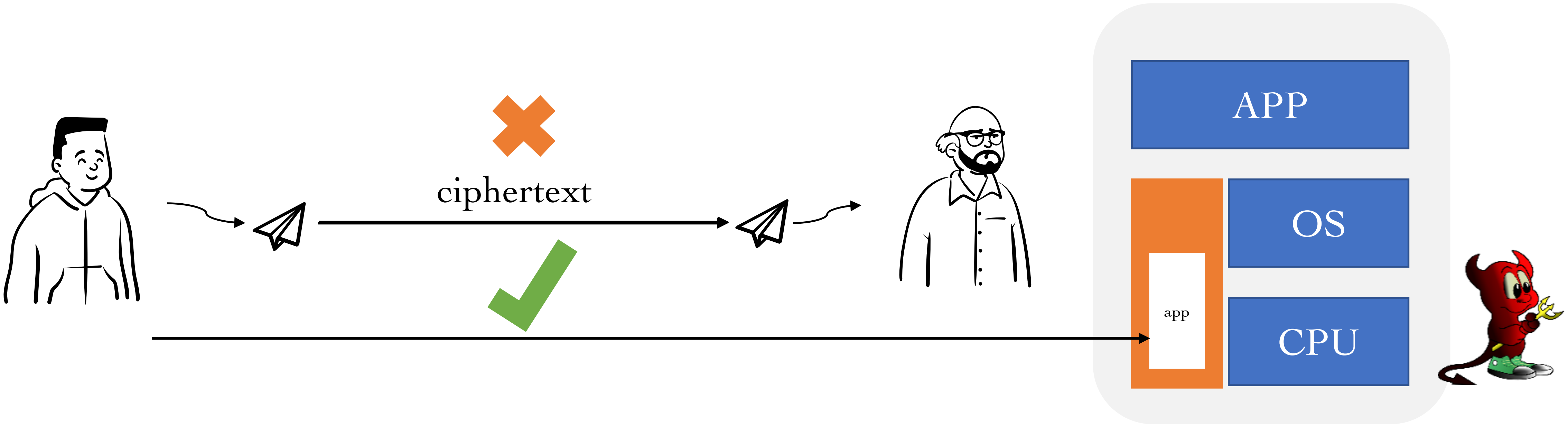
https://rujia.uk/

Abstraction

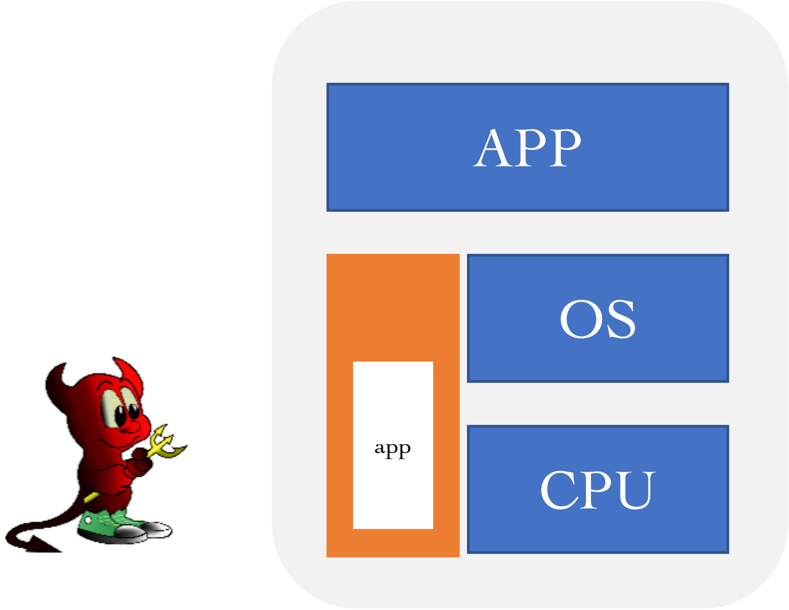


The attacking surface is large

Needs on the Hardware Protection



Trusted Execution Environment



Hardware support for

- Isolated execution: Isolated Execution Environment
- Ability to convince remote verifiers: (Remote) Attestation
- Protected storage: Sealing

TEE Solutions

X86 architecture:

- Intel SGX
- AMD SEV

Arm architecture:

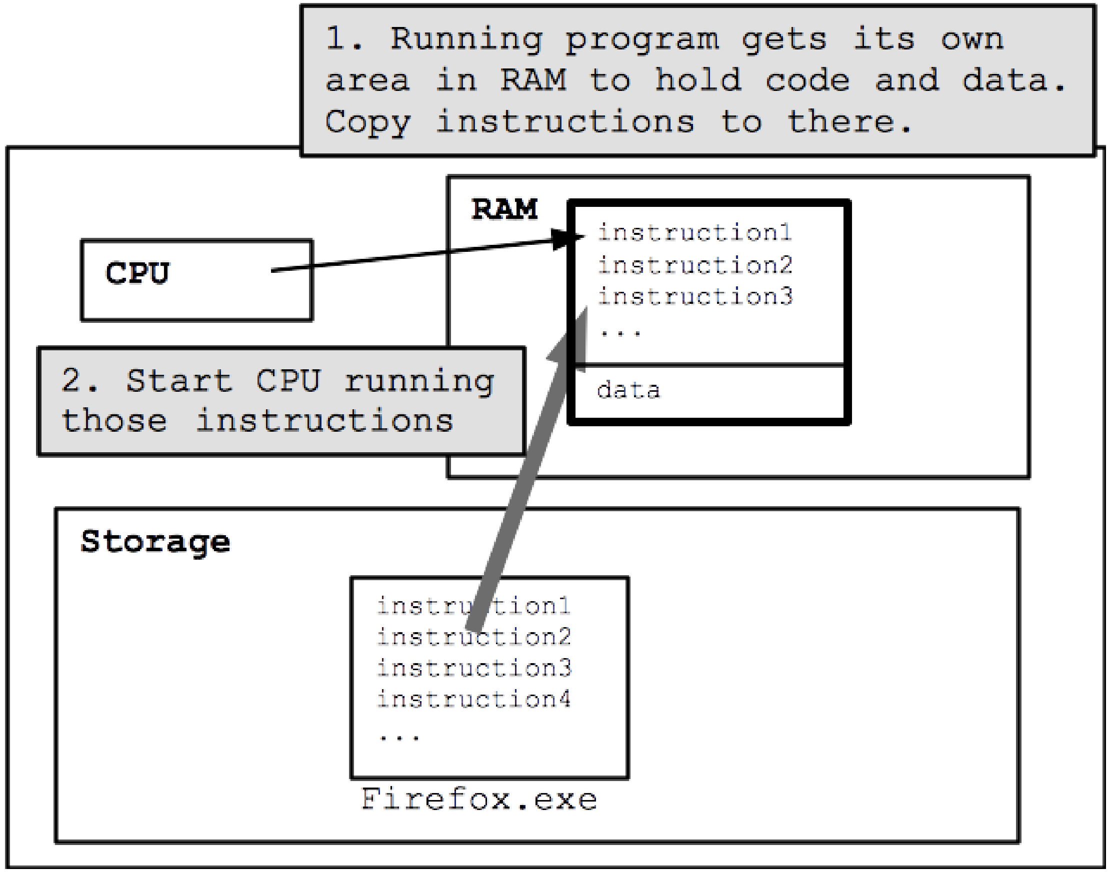
- OP-TEE
- iTrustee

RISC-V architecture:

- Keystone
- Sanctum

Isolated Execution

How Does a Program Run



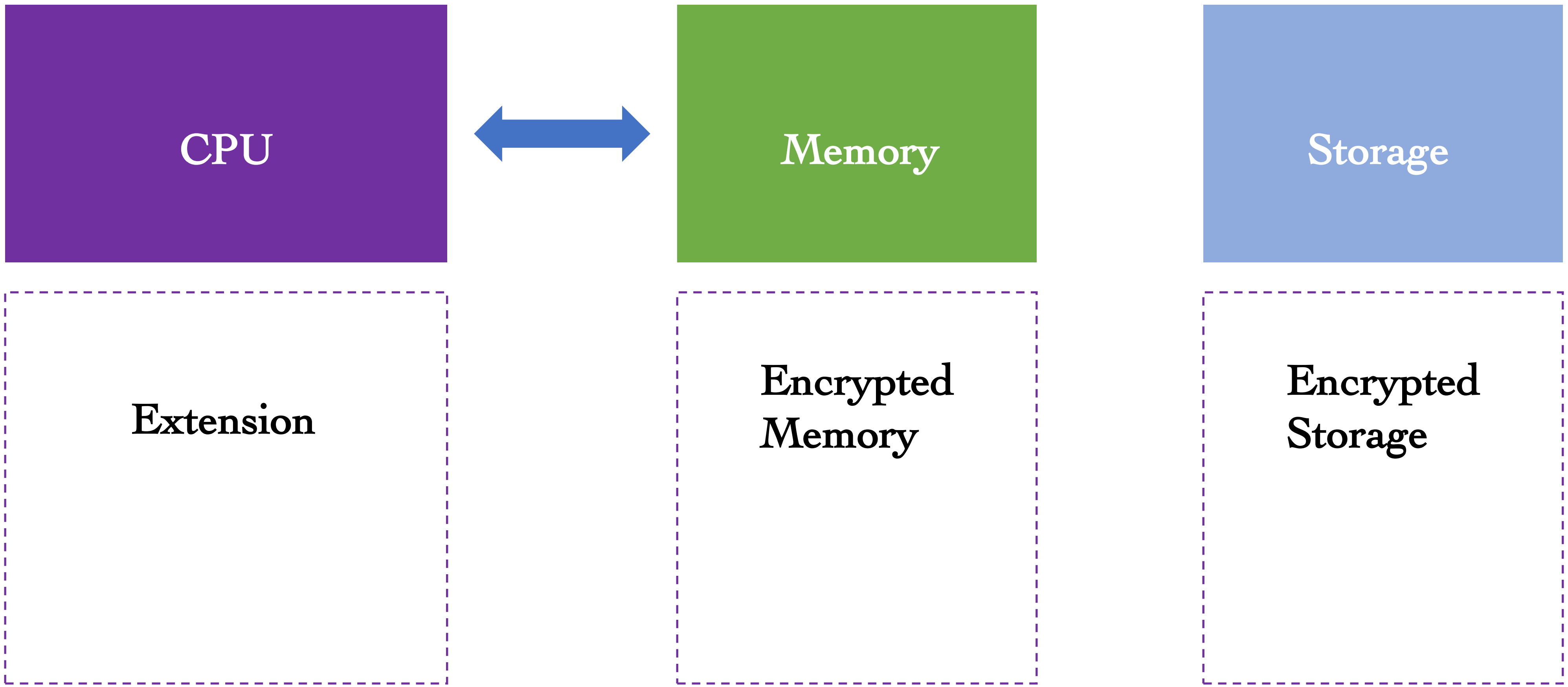
CPU runs a *fetch/execute cycle*

- fetch one instruction in sequence
- execute (run) that instruction, e.g. do the addition
- fetch the next instruction, and so on

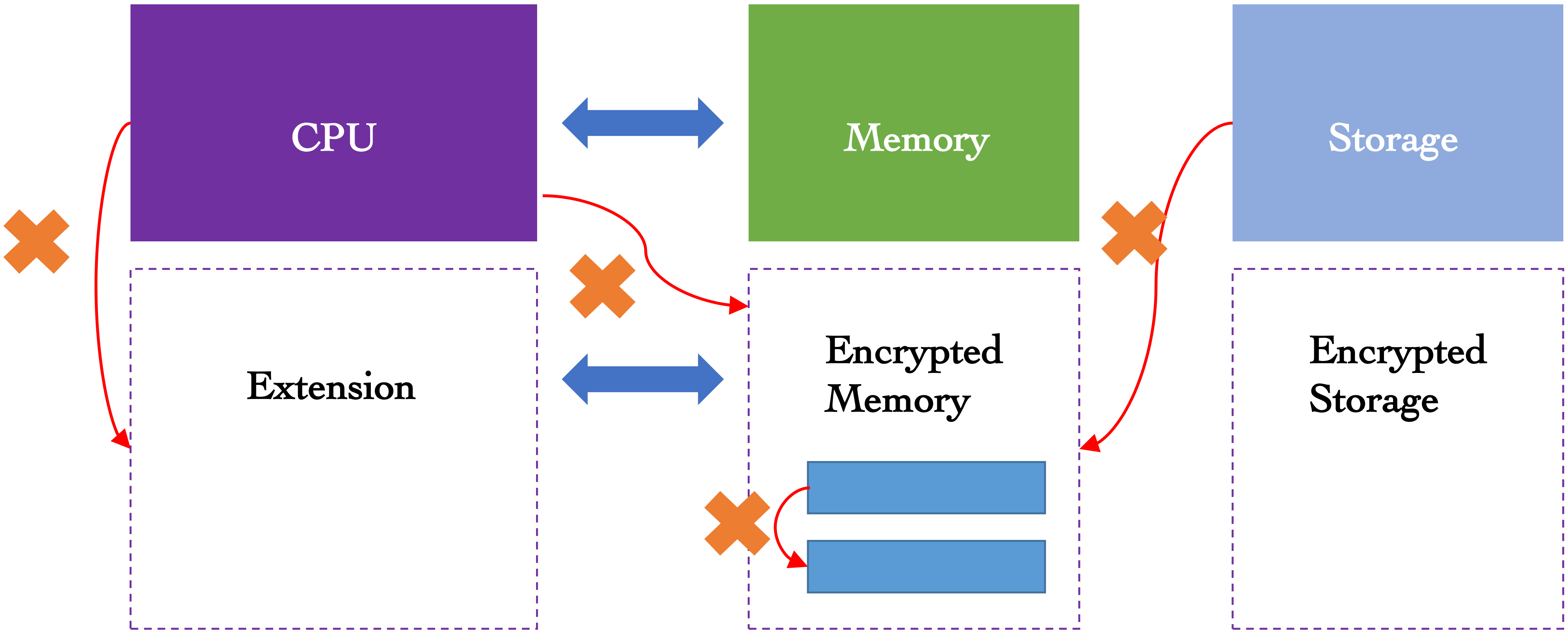
https://web.stanford.edu/class/cs101/software-1.html

How Does a Program Run

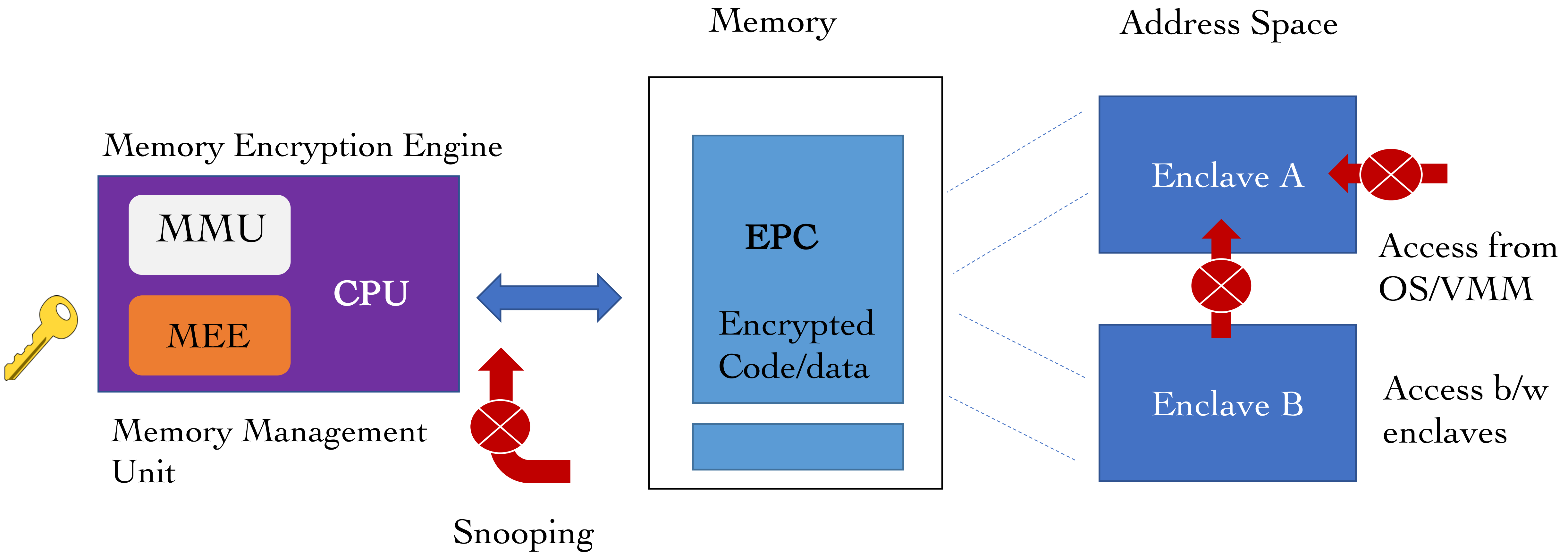
Trusted Execution Environment



Trusted Execution Environment



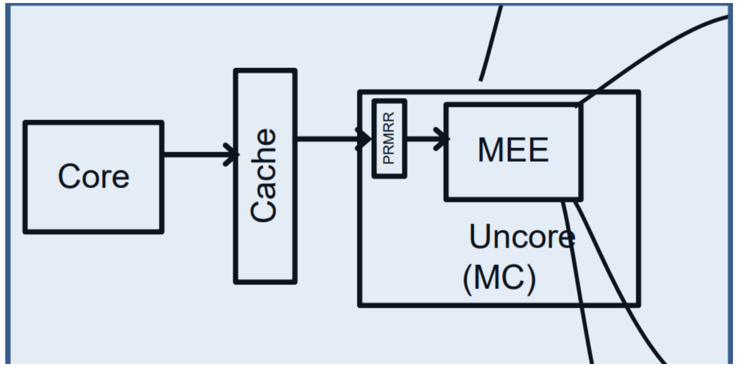
Intel SGX Architecture



Memory Encryption Engine

SGX cryptographic protection of memory is supported by the Memory Encryption Engine

Hardware unit - extension of the Memory Controller

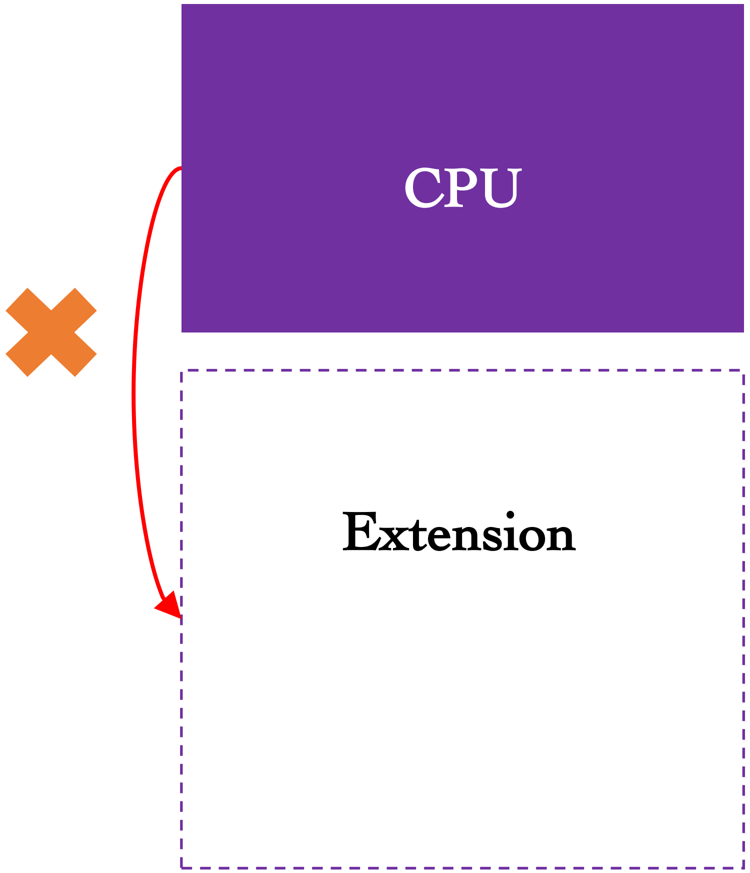


- Objectives:
  - – Data Confidentiality: Collections of memory images of DATA written to the DRAM (into different addresses and points in time) cannot be distinguished from random data.
  - – Integrity: DATA read back from DRAM to LLC is the same DATA that was most recently written from LLC to DRAM.

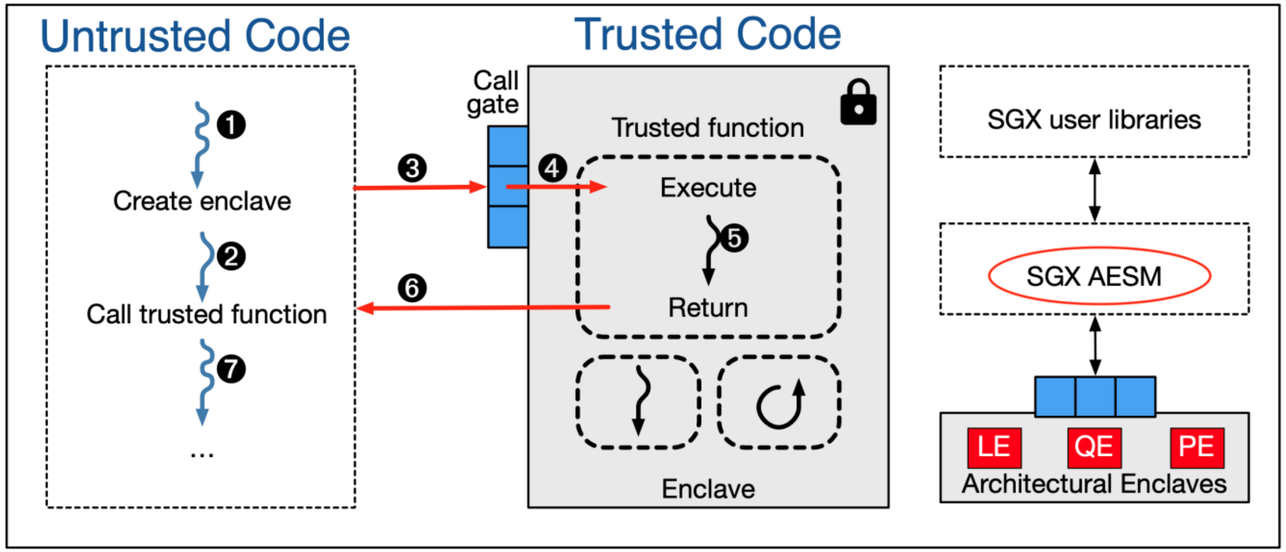
- Keys are randomly generated at reset by a HW DRNG module.
- Accessible only to MEE hardware

Isolated Execution

- **SGX introduces notion of enclave**
  - Isolated memory region for code & data
  - New CPU instructions to manipulate enclaves
    and new enclave execution mode
- **Enclave memory encrypted and integrity-protected by hardware**
  - Memory encryption engine (MEE)
  - No plaintext secrets in main memory



Isolated Execution



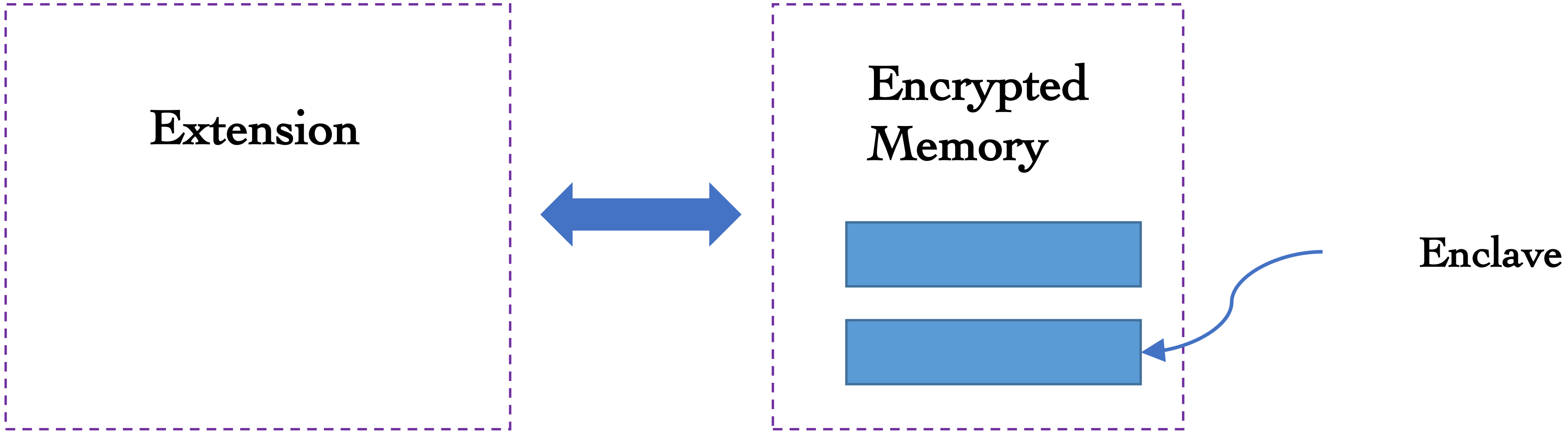
- **Enclave memory can be accessed only by enclave code**
  - Protection from privileged code (OS, hypervisor)

Application has ability to defend secrets

1. Attack surface reduced to just enclaves and CPU
2. Compromised software cannot steal application secrets
3. Protects confidentiality and integrity of code & data in untrusted environments
4. Platform owner considered malicious
5. Only CPU chip and isolated region trusted

Intel SGX Instruction

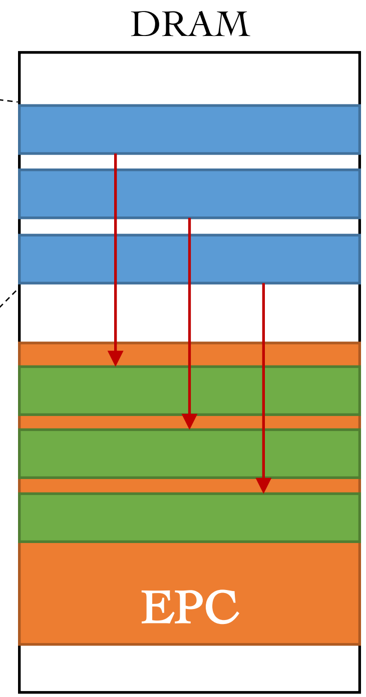
Super.	Description	User	Description
EADD	Add a page	EENTER	Enter an enclave
EBLOCK	Block an EPC page	EEXIT	Exit an enclave
ECREATE	Create an enclave	EGETKEY	Create a cryptographic key
EDBGRD	Read data by debugger	EREPORT	Create a cryptographic report
EBDGWR	Write data by debugger	ERESUME	Re-enter an enclave



Enclave Construction

```
1 { char input_buf[BUFFER_SIZE];
2 { char output_buf[BUFFER_SIZE];

    int process_request(char *in, char *out)
    {
      copy_msg(in, input_buf);
      if(verify_MAC(input_buf))
      {
        decrypt_msg(input_buf);
3       process_msg(input_buf, output_buf);
        encrypt_msg(output_buf);
        copy_msg(output_buf, out);
        EEXIT(0);
      } else
        EEXIT(-1);
    }
```



EPC is limited (only ~94MB available)

Enclave Page Cache (EPC)

Enclave populated using special instruction (EADD)

- Contents initially in untrusted memory
- Copied into EPC in 4KB pages

Both data & code copied before execution commences in enclave

Code and Workflow

SGX application: untrusted code

```
char request_buf[BUFFER_SIZE];
char response_buf[BUFFER_SIZE];

int main()
{
 ...
 while(1)
 {
  receive(request_buf);
  ret = EENTER(request_buf, response_buf);
  if (ret < 0)
   fprintf(stderr, "Corrupted message\n");
  else
   send(response_buf);
 }
 ...
}
```

Enclave: trusted code

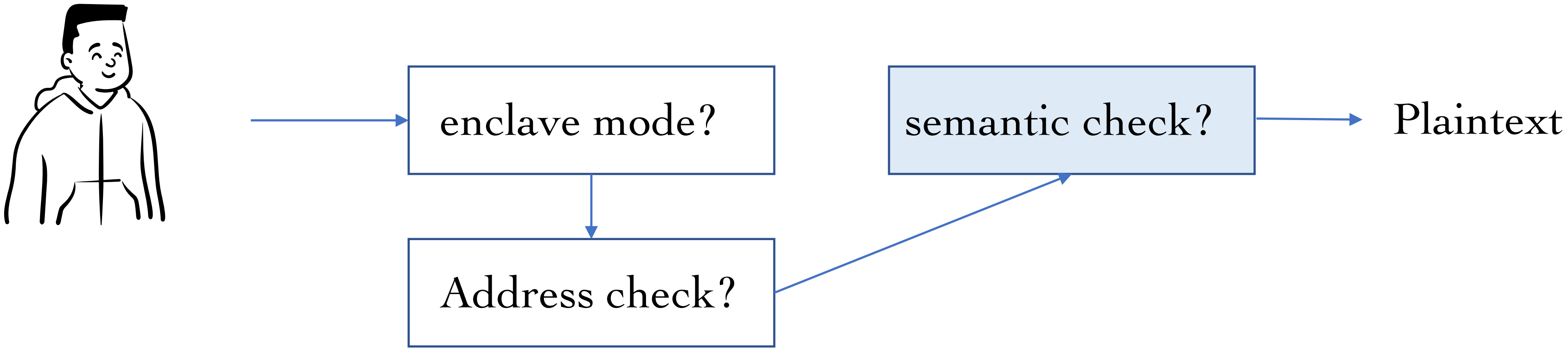
```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
 copy_msg(in, input_buf);
 if(verify_MAC(input_buf))
 {
  decrypt_msg(input_buf);
  process_msg(input_buf, output_buf);
  encrypt_msg(output_buf);
  copy_msg(output_buf, out);
  EEXIT(0);
 } else
  EEXIT(-1);
}
```

Server:

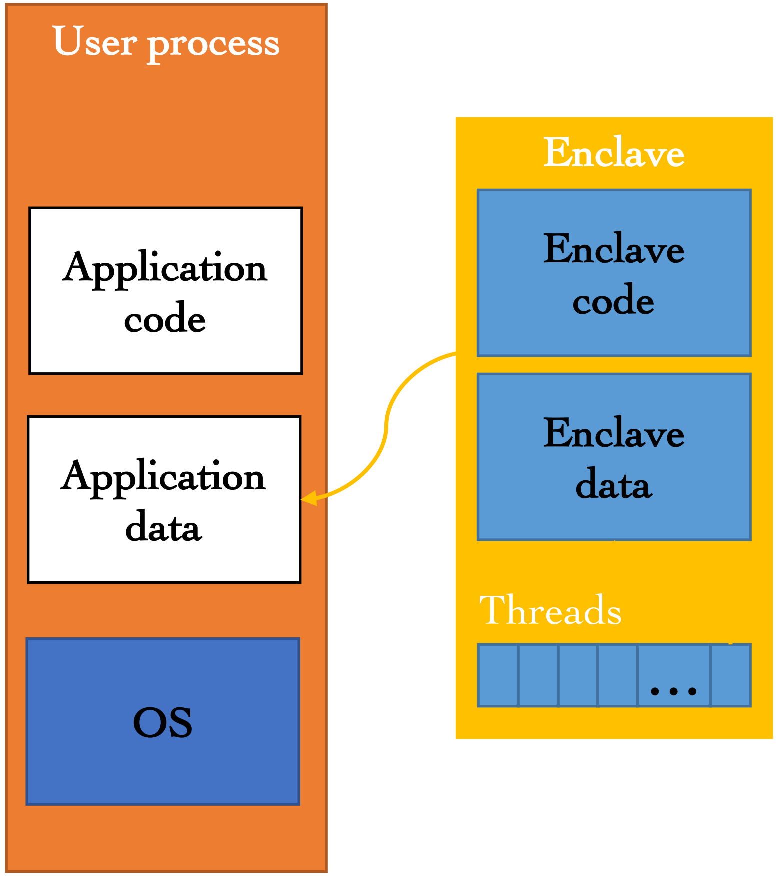
- Receives encrypted requests
- Processes them in enclave
- Sends encrypted responses

CPU-level Access Control




```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

Isolated Execution Summary



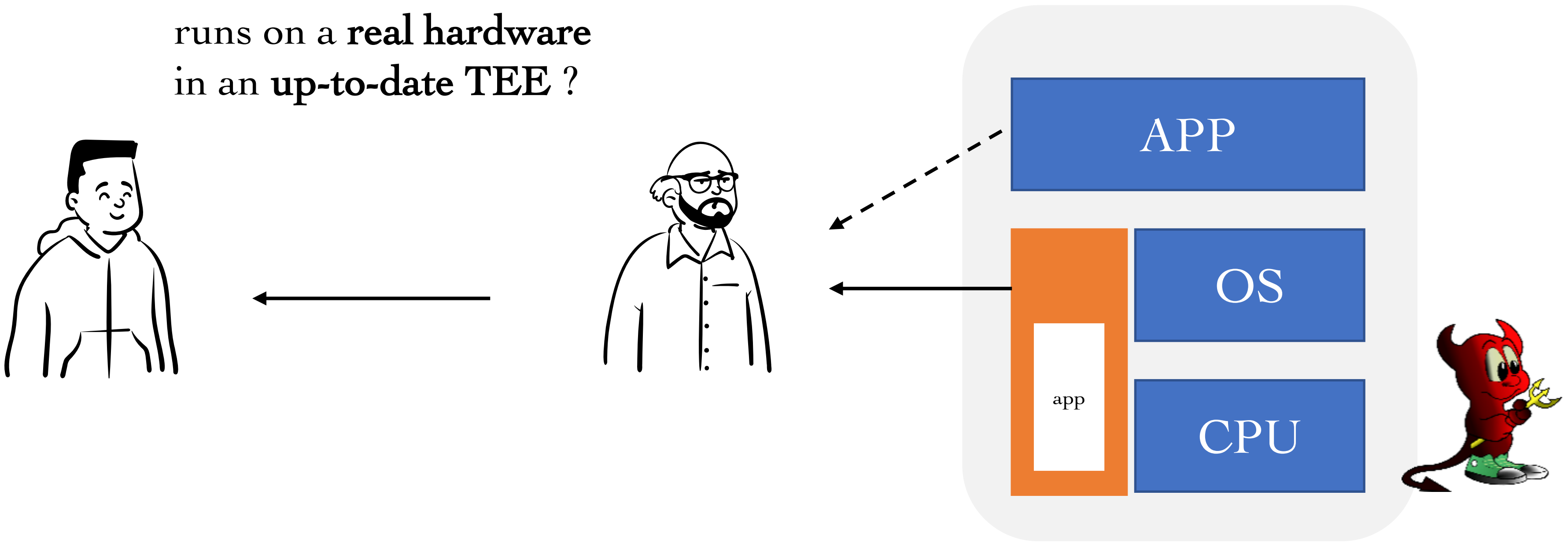
Trusted execution environment (TEE) in process

- Own code & data
- Controlled entry points (Access control)
- Provides confidentiality & integrity
- Supports multiple threads
- Full access to application memory

Attestation

Remote Attestation

runs on a **real hardware**
in an **up-to-date TEE** ?

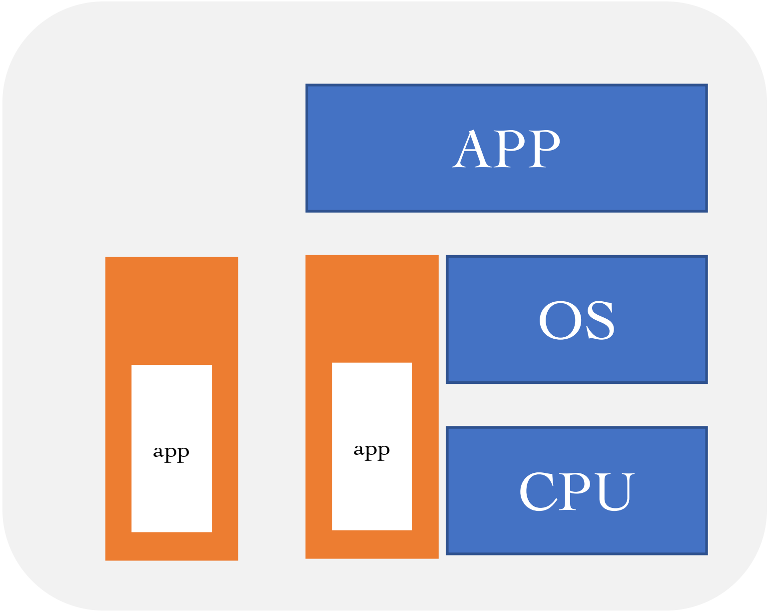


Is code **really** running inside an SGX enclave?

Local Attestation

Local attestation
	Prove enclave's identity (= measurement) to another enclave on same CPU

Attestation is a mechanism to verify that the application runs on a real hardware in an up-to-date TEE with the expected initial state.



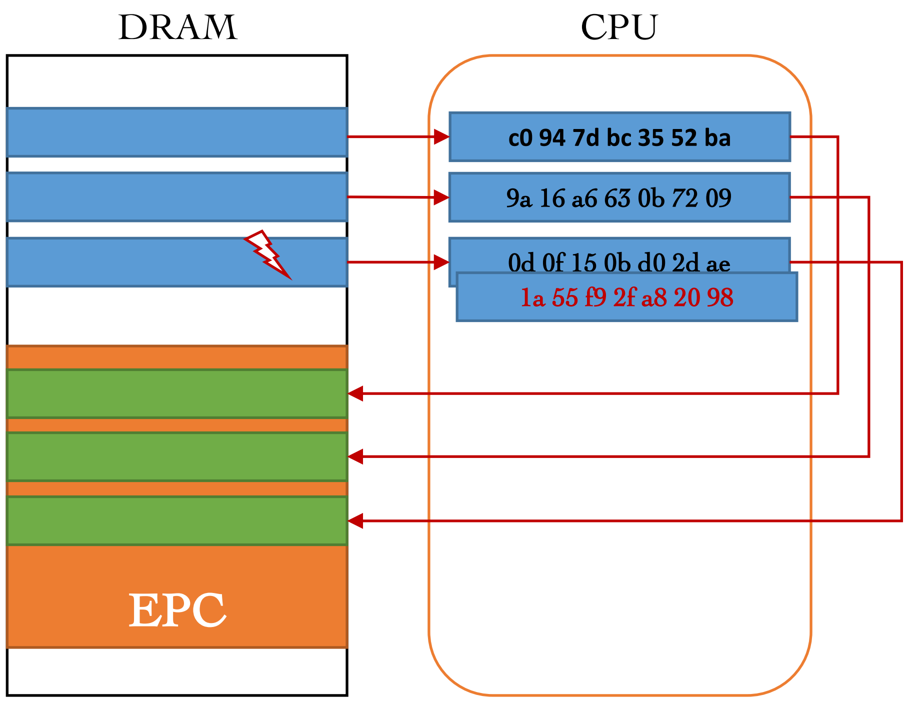
TEE Measurement

- Enclave contents distributed in plaintext
  - Must not contain any (plaintext) confidential data
- Secrets provisioned after enclave constructed and integrity verified
- Problem: what if someone tampers with enclave?
  - Contents initially in untrusted memory

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

```
int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    copy_msg(output_buf, external_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

TEE Measurement



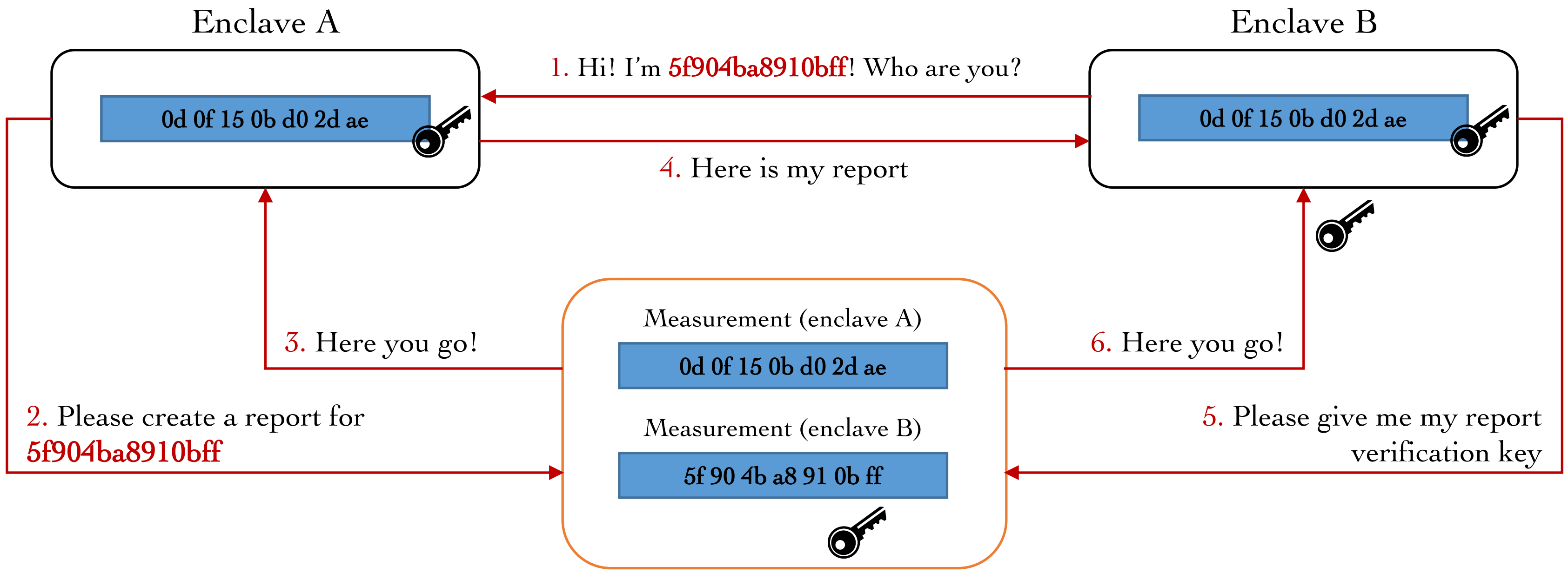
- CPU calculates enclave measurement hash during enclave construction
  - Each new page extends hash with page content and attributes (read/write/execute)
  - Hash computed with SHA-256

- Measurement can be used to attest enclave to local or remote entity

CPU calculates enclave measurement hash during enclave construction
Different measurement if enclave modified

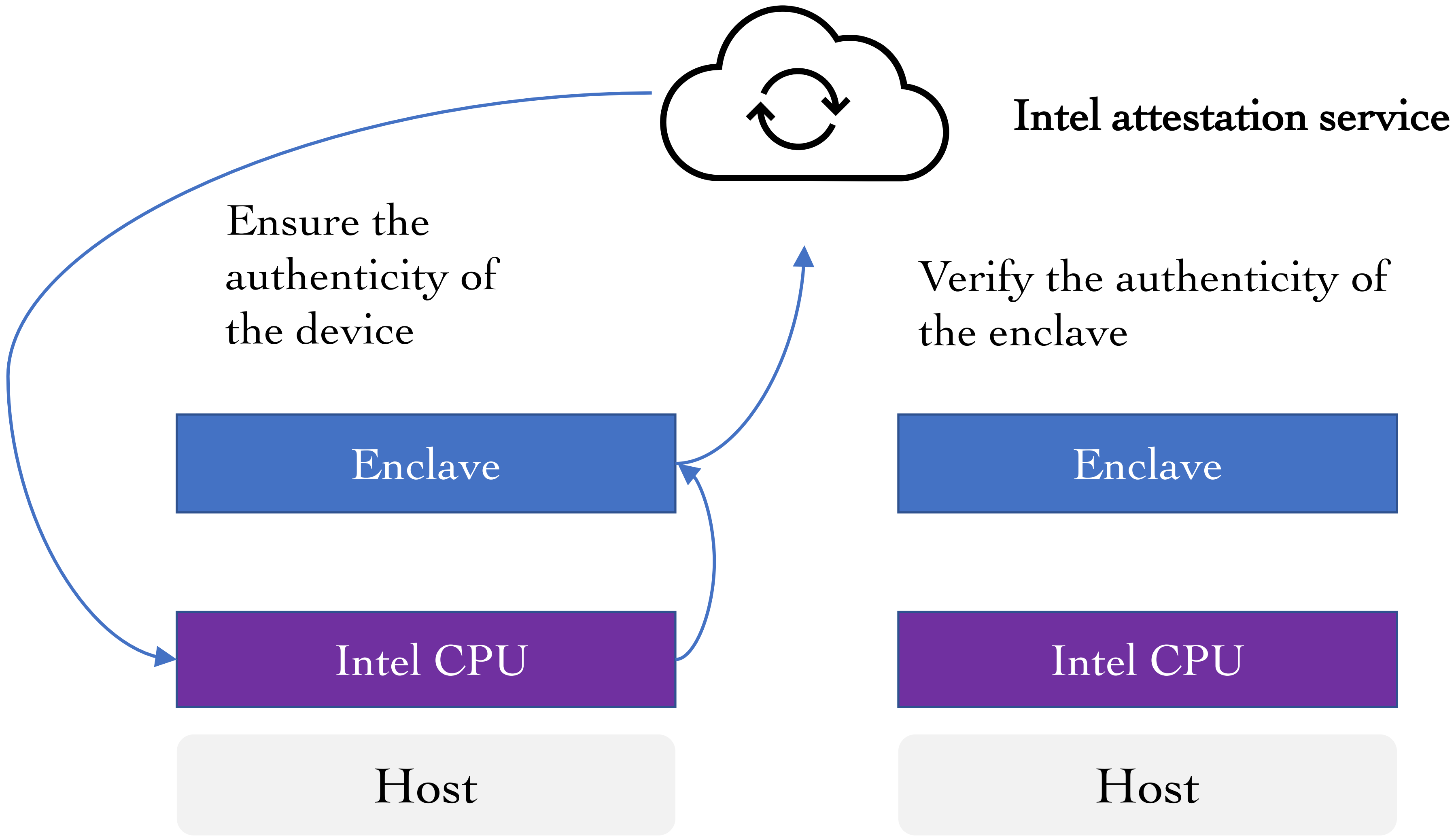
Local Attestation

- Prove identity of A to local enclave B



1. Target enclave B measurement required for **key generation**
2. Report contains information about target enclave B, including its measurement
3. CPU fills in report and creates MAC using report key, which depends on random CPU fuses and target enclave B measurement
4. Report sent back to target enclave B
5. Verify report by CPU to check that generated on same platform, i.e. MAC created with same report key (available only on same CPU)
6. Check MAC received with report and do not trust A upon mismatch

Remote Attestation Overview



Report and Quote

```
char input_buf[BUFFER_SIZE];
char output_buf[BUFFER_SIZE];

int process_request(char *in, char *out)
{
  copy_msg(in, input_buf);
  if(verify_MAC(input_buf))
  {
    decrypt_msg(input_buf);
    process_msg(input_buf, output_buf);
    encrypt_msg(output_buf);
    copy_msg(output_buf, out);
    EEXIT(0);
  } else
    EEXIT(-1);
}
```

Report

Measurement

REPORTDATA (output)

Quote

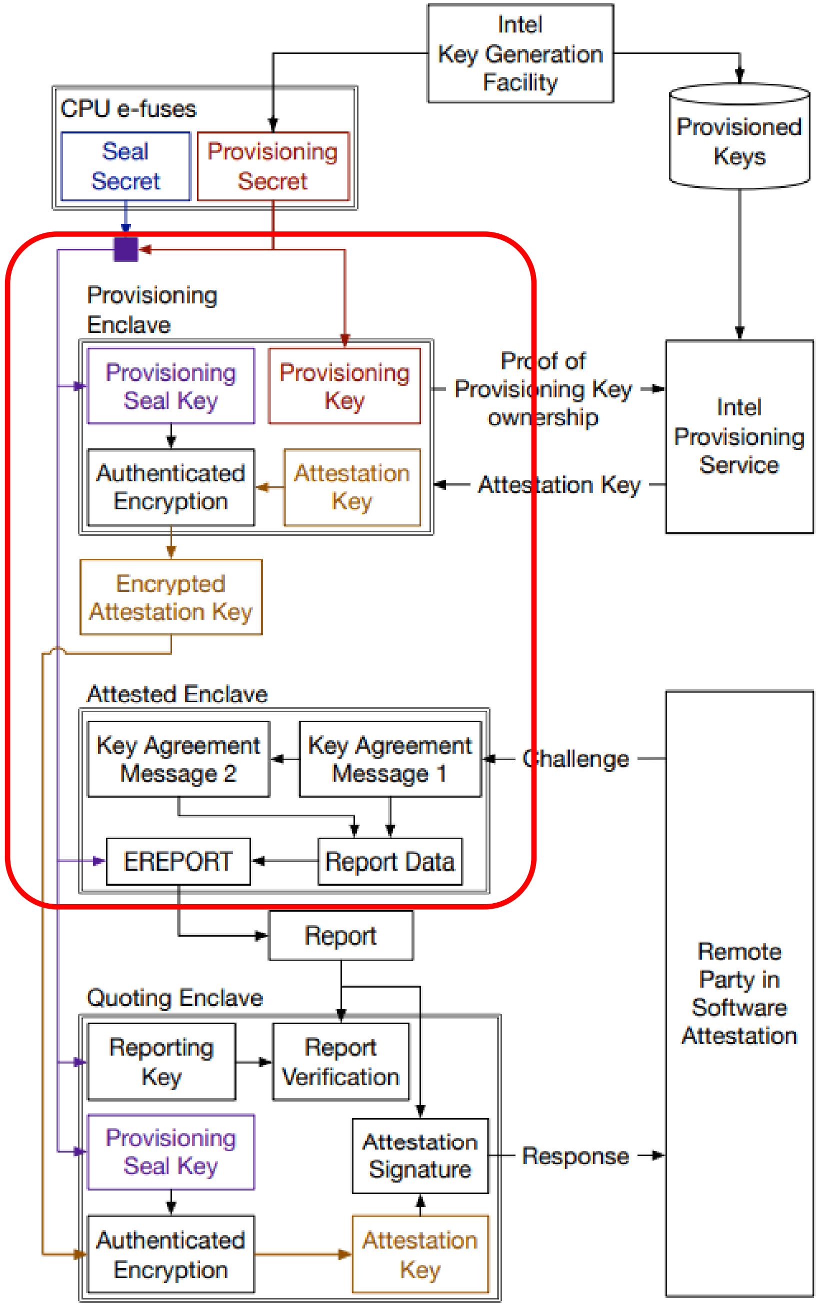
Signature

Measurement

REPORTDATA (output)

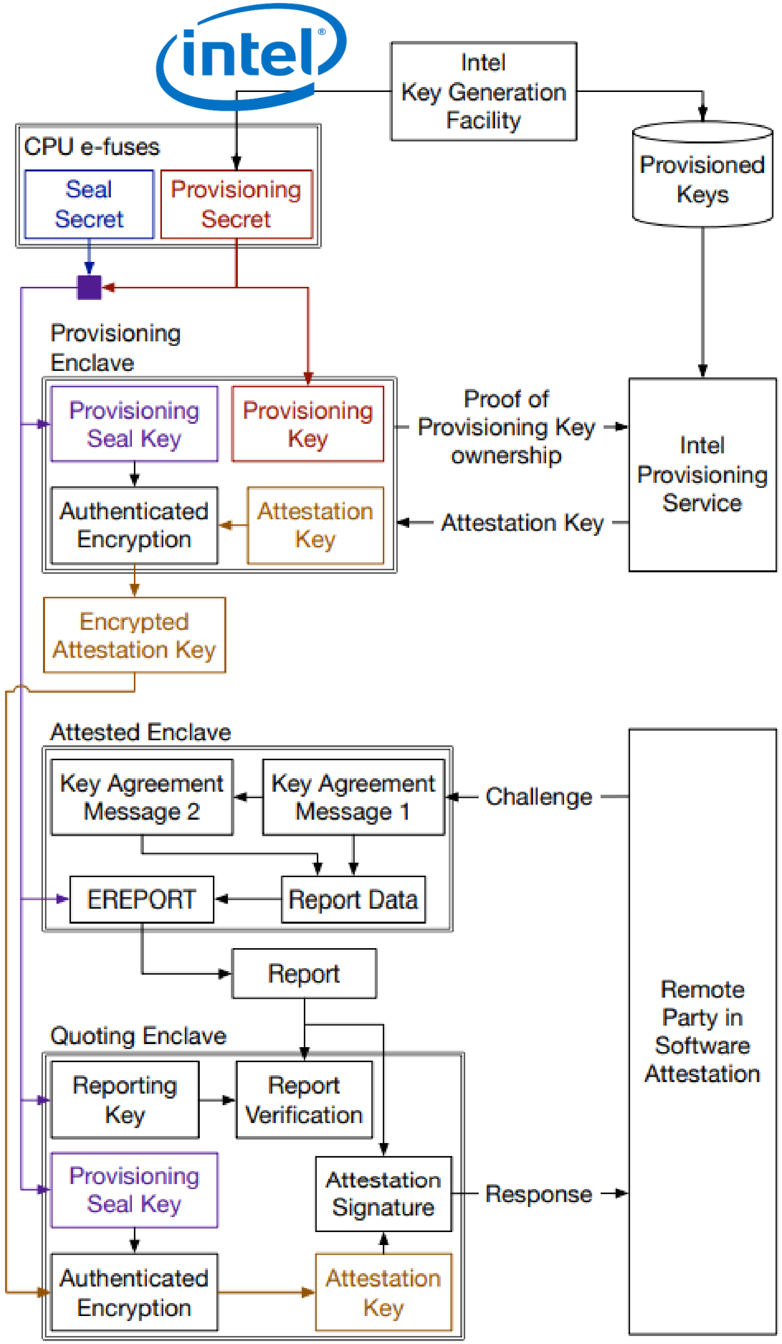
PE and QE

- Transform local report to remotely verifiable “quote”
- Based on provisioning enclave (PE) and quoting enclave (QE)
  - Architectural enclaves provided by Intel
  - Execute locally on user platform
- Each SGX-enabled CPU has unique key fused during manufacturing
  - Intel maintains database of keys



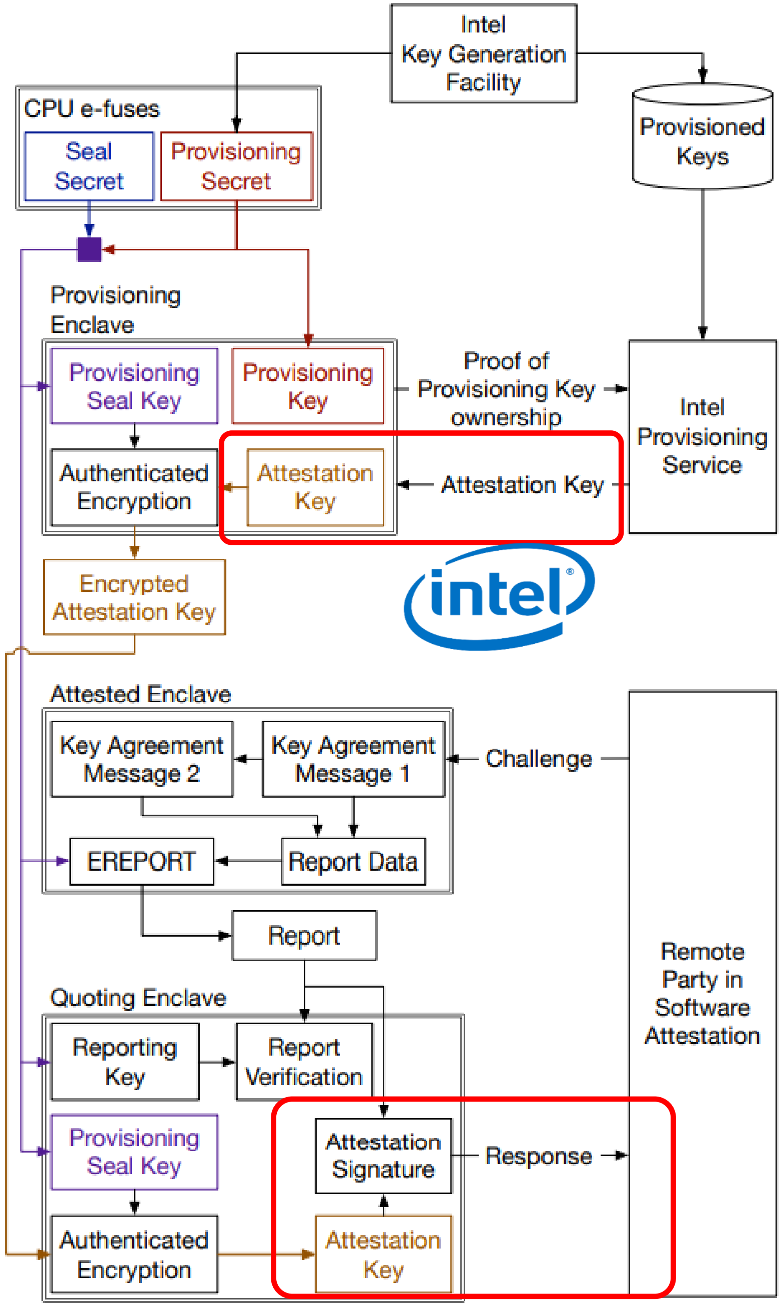
Root Provisioning Key

- The first fused key created by Intel at manufacturing time, is the Root Provisioning Key (RPK)
- Intel is also responsible for maintaining a database of all keys ever produced



Remote Attestation

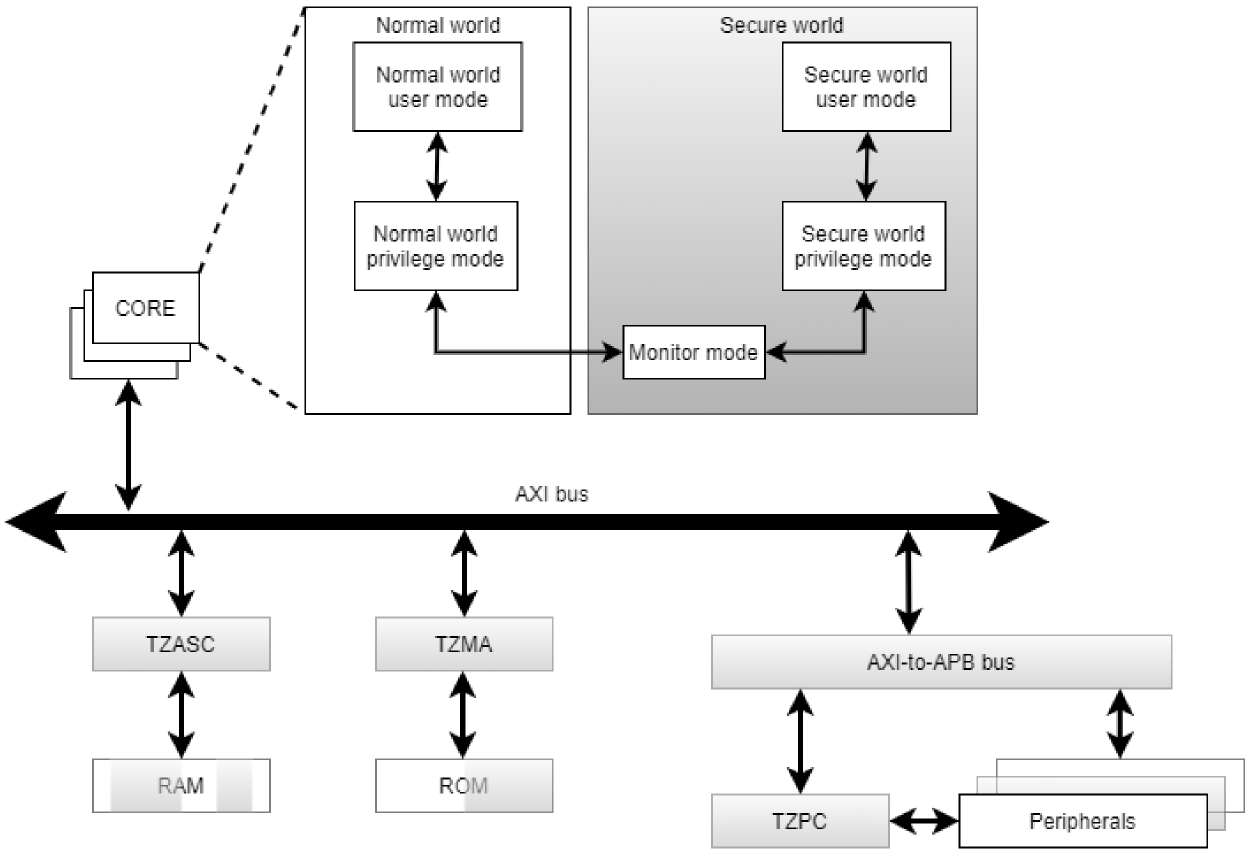
- PE communicates with Intel attestation service
  - Proves it has key installed by Intel
  - Receives asymmetric attestation key
- QE performs local attestation for enclave
  - QE verifies report and signs it using attestation key
  - Creates quote that can be verified outside platform
- Quote and signature sent to remote attester, which communicates with Intel attestation service to verify quote validity



Arm Trust Zone

- TrustZone is a set of hardware security extensions in Arm processors.
    - **Introduced in application processors (Cortex-A) in 2004**
    - Introduced in microcontrollers (Cortex-M) in 2016

- Focusing on the extensions in Cortex-A, due to its:
    - widely deployment,
    - available documents,
    - growing interest from academia.



Privacy Concerns on Attestation

Attestation using standard asymmetric signing schemes has drawn some privacy concerns

Quote 1

I know you, and now I can identify/trace you !

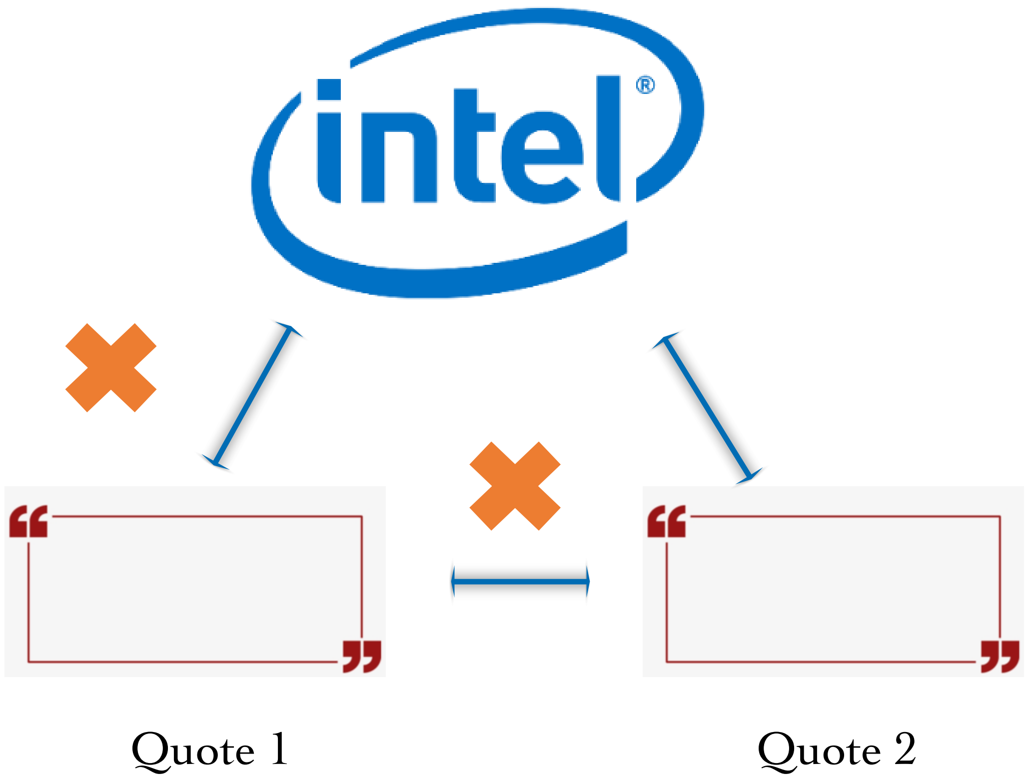
Intel Enhanced Privacy ID (EPID)

Attestation using standard asymmetric signing schemes has drawn some privacy concerns

- EPID is a type of group signature scheme that allows a platform to sign objects without uniquely identifying the platform or linking different signatures.

- Each EPID signer belongs to a "group", and verifiers use the group's public key to verify signatures.

- A typical size for a fully populated group is a million to a few million platforms.



EPID scheme Join protocol

- EPID is a type of group signature scheme that allows a platform to sign objects without uniquely identifying the platform or linking different signatures.

- A typical size for a fully populated group is a million to a few million platforms.

– $\texttt{Setup}(1^k)$: on input a security parameter 1^k, this algorithm returns the public parameters pp of the system.
– $\texttt{GKeygen}(pp)$: on input the public parameters pp, this algorithm generates the issuer's key pair $(\mathsf{isk}, \mathsf{ipk})$. We assume that ipk contains pp and so we remove pp from the inputs of all following algorithms.
– $\texttt{Join}$: this is an interactive protocol between a platform $\mathcal{P}$, taking as inputs ipk, and the issuer $\mathcal{I}$ owning isk. At the end of the protocol, the platform returns either $\perp$ or a signing key sk whereas the issuer does not return anything.
– $\texttt{KeyRevoke}(\{\mathsf{sk}_i\}_{i=1}^m)$: this algorithm takes as input a set of m platform secret keys sk_i and returns a corresponding key revocation list $\texttt{KRL}$ containing m elements that will be denoted as $\texttt{KRL}[i]$, for $i \in [1, m]$.
– $\texttt{SigRevoke}(\{(\mu_i)\}_{i=1}^n)$: this algorithm takes as input a set of n EPID signatures $\{(\mu_i)\}_{i=1}^n$ and returns a corresponding signature revocation list $\texttt{SRL}$ containing n elements that will be denoted as $\texttt{SRL}[i]$, for $i \in [1, n]$.
– $\texttt{Sign}(\mathsf{ipk}, \mathsf{sk}, \mathsf{m}, \texttt{SRL})$: this algorithm takes as input the issuer's public key ipk, a platform secret key sk a message m and a signature revocation list $\texttt{SRL}$ and returns an EPID signature μ.

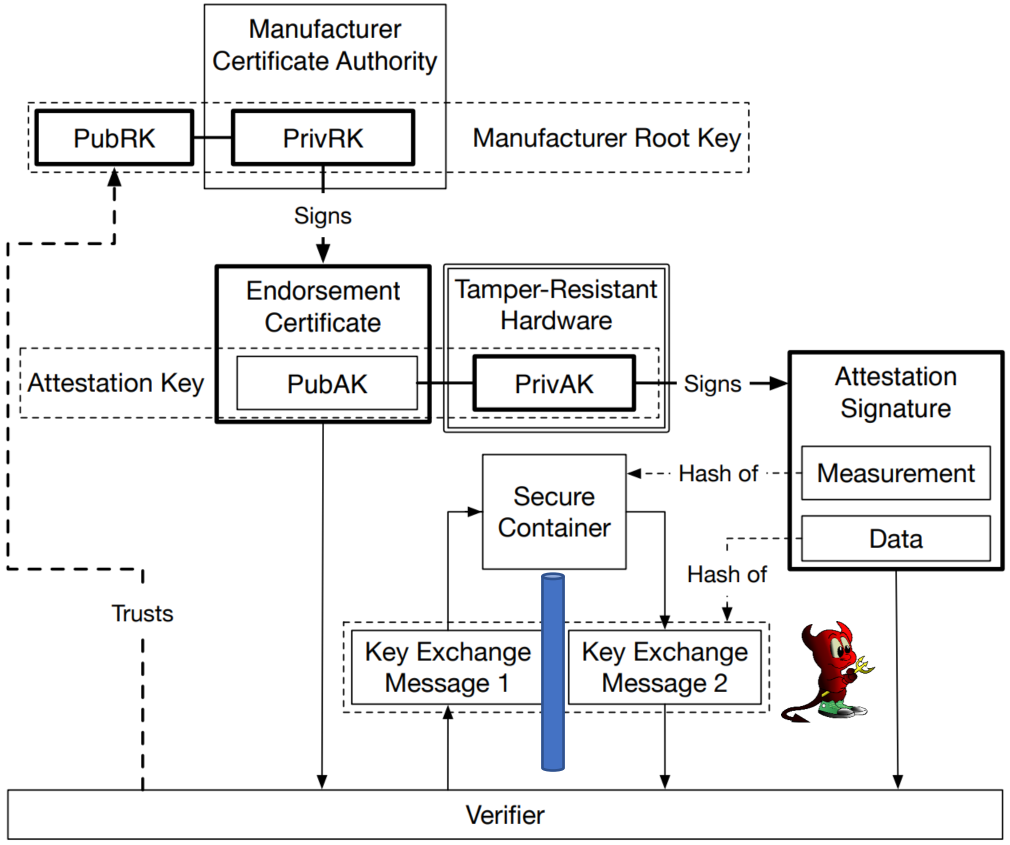
Sanders, Olivier, and Jacques Traoré. "EPID with Malicious Revocation." In Cryptographers' Track at the RSA Conference, pp. 177-200. Springer, Cham, 2021.

Use Case 1: Authenticated Key Agreement

1. The verifier starts executing the key exchange protocol and sends the first message g^A to the software inside the secure container.

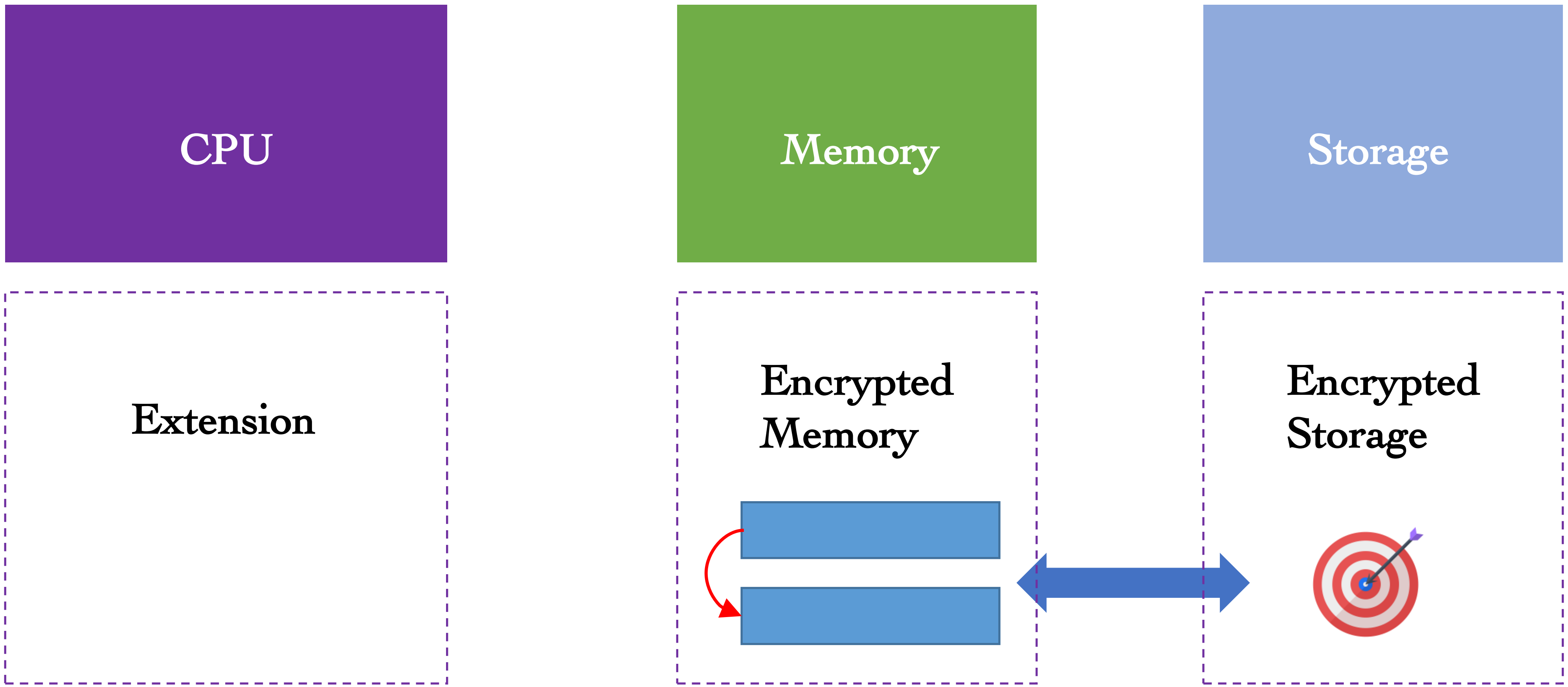
2. The software inside the container produces the second key exchange message, g^B, and asks the trusted hardware to attest the cryptographic hash of both key exchange messages, h(g^A||g^B).

3. The verifier receives the second key exchange and attestation signature, and authenticates the software inside the secure container by checking all the signatures

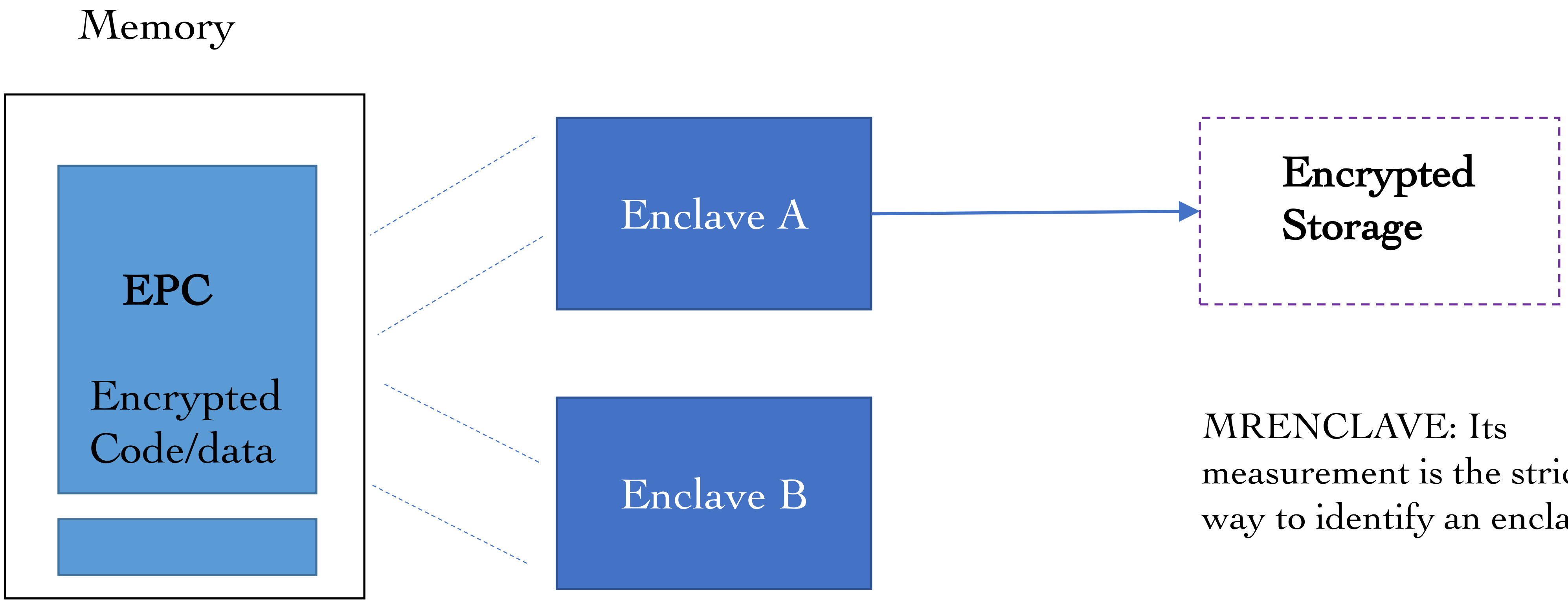


Sealing

Encrypted Storage

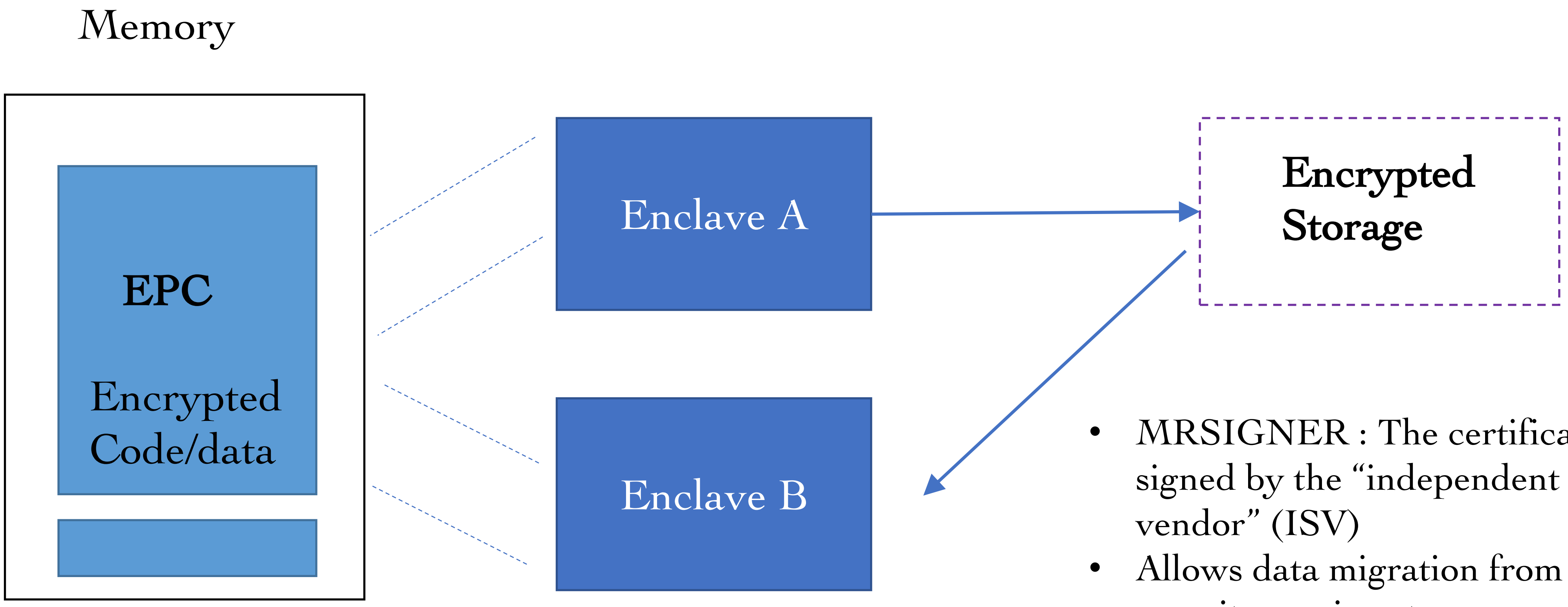


MRENCLAVE



MRENCLAVE: Its measurement is the strictest way to identify an enclave.

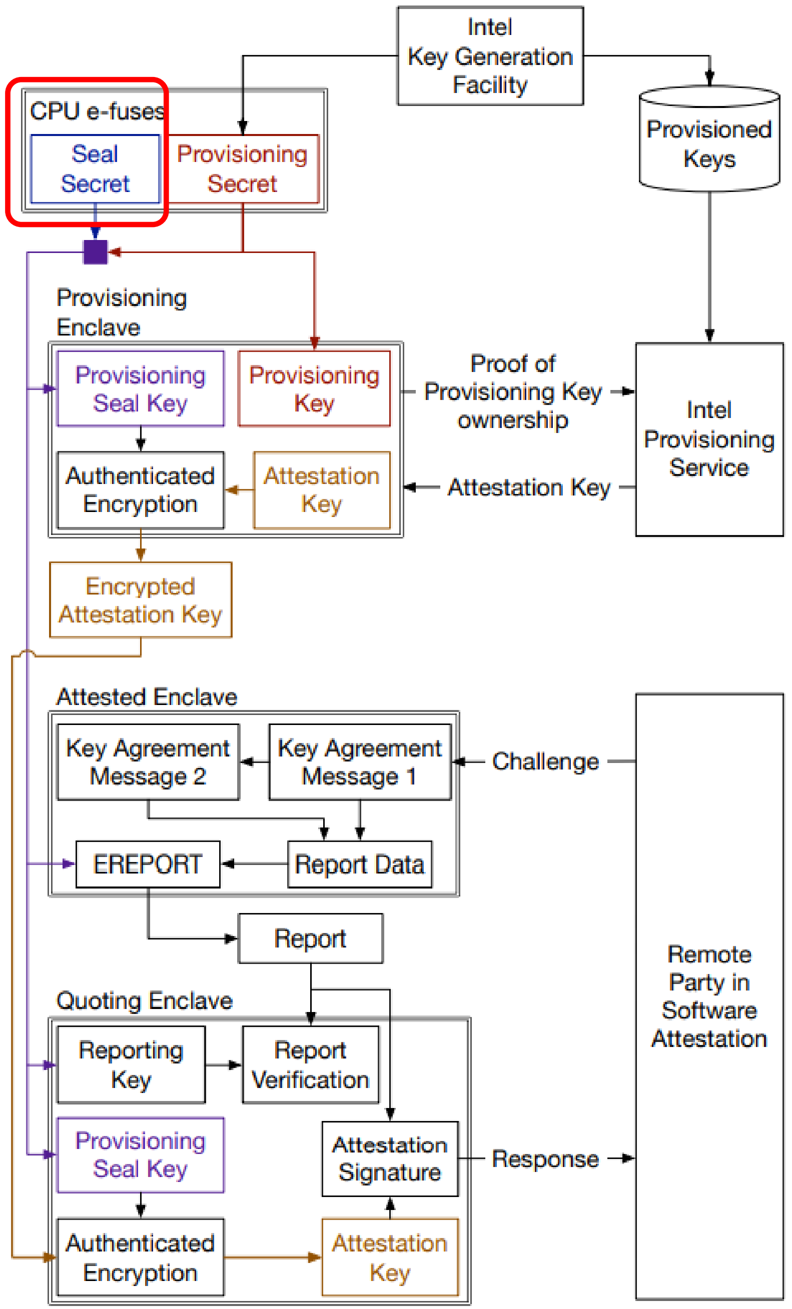
MRSIGNER



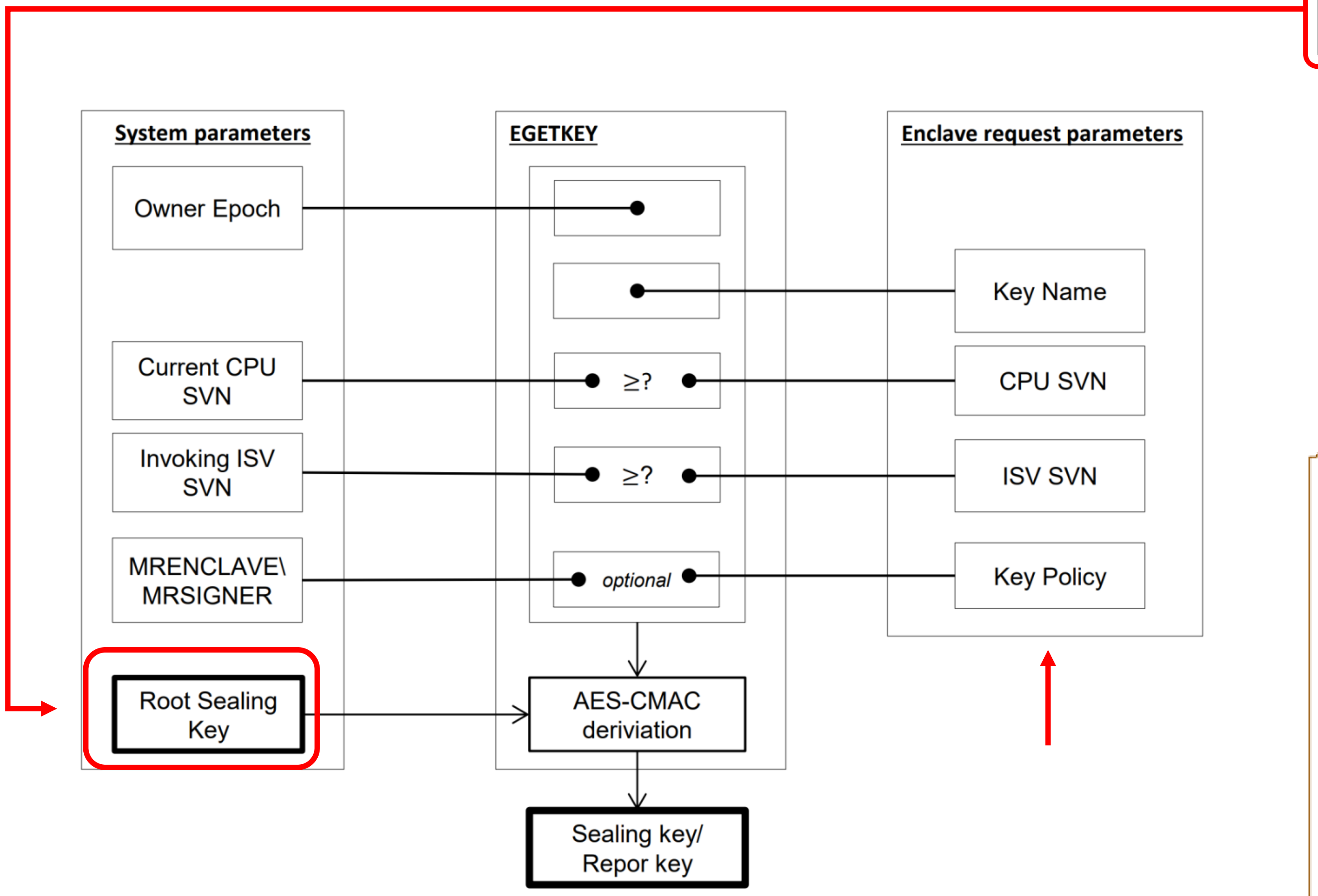
- MRSIGNER : The certificate is signed by the “independent software vendor” (ISV)
- Allows data migration from old security versions to new ones

Root Provisioning Key

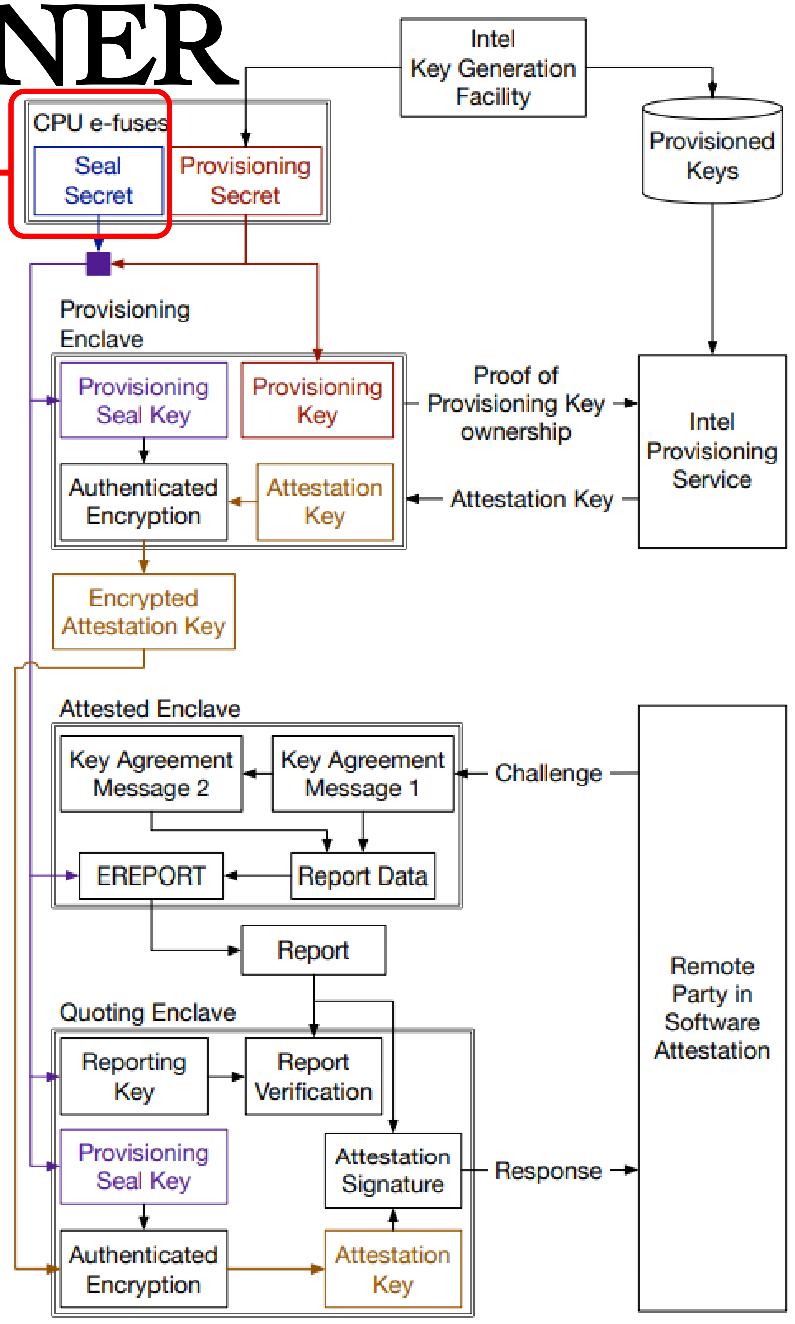
- each platform should assume that its RSK value is both unique and known only to itself.



MRENCLAVE and MRSIGNER



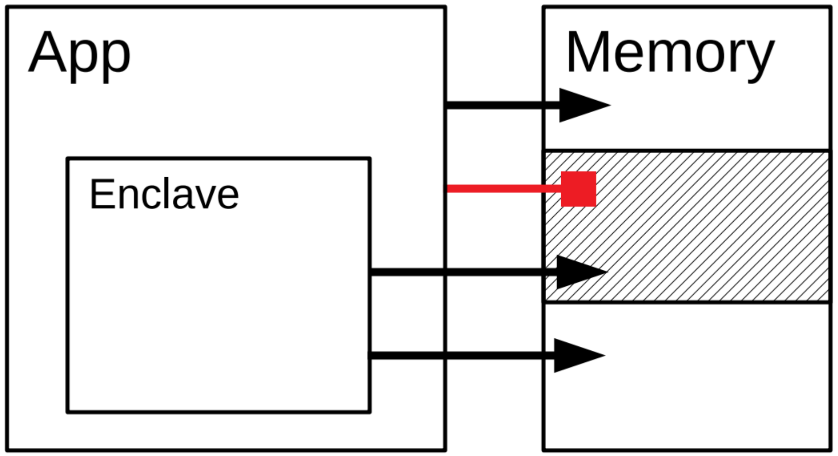
Simplified EGETKEY derivation process



Summary

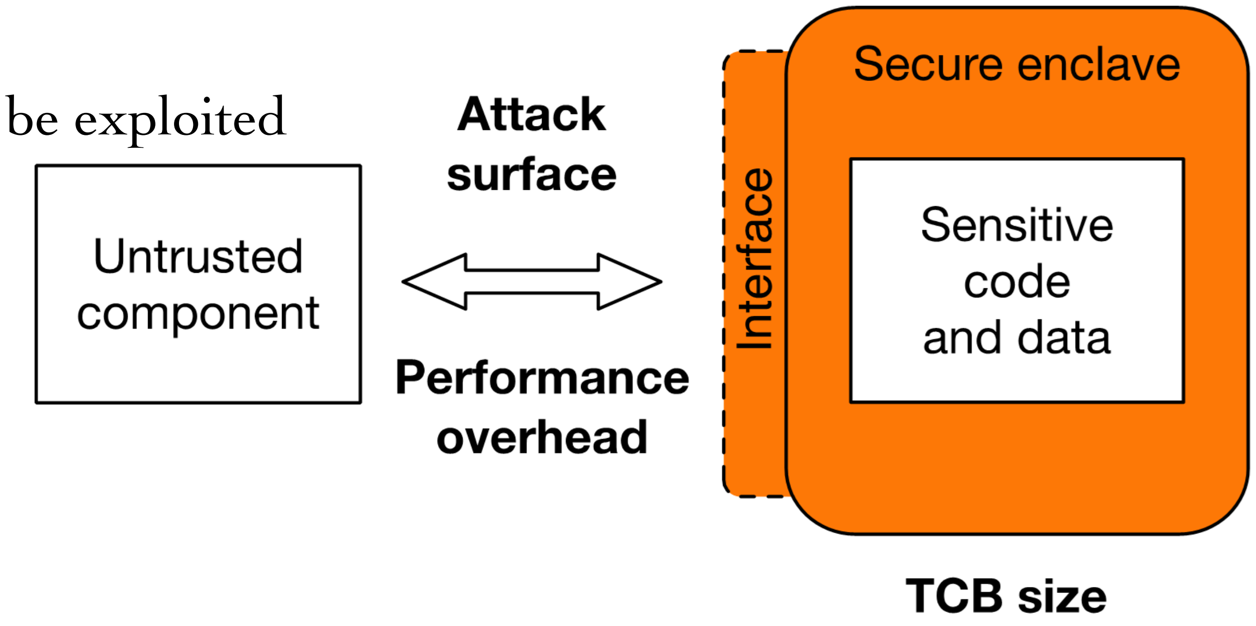
What an Enclave Can Do?

- Computations
- Access its own [encrypted] memory
- Access app memory
- Communicate with user, but insecurely
- Communicate with another party, which can be secure if the enclave shares a key with the other party
- Attest its identity (a hash of its binary and initial data) to another party
- "Seal" data, i.e. encrypt data with a key that only it can access, for persistent storage – Can use Platform Service Enclave (PSE) for trusted time and monotonic counter
- Teardown



SGX Limitations & Research Challenges

- Amount of memory enclave can use needs to be known in advance
  - Dynamic memory support in SGX v2
- Security guarantees not perfect
  - Vulnerabilities within enclave can still be exploited
  - Side-channel attacks possible
- Performance overhead
  - Enclave entry/exit costly
  - Paging very expensive
- Application partitioning? Legacy code? …



Reference

- Intro to Intel SGX , **Mark D. Ryan**
- Attacks and Defenses for Intel SGX, **Taesoo Kim**
- Intel SGX Explained, **Victor Costan and Srinivas Devadas**
- Privacy-Preserving Analytics in and out of the Clouds, **Jon Crowcroft**