TEL AVIV UNIVERSITY

Information Security – Theory vs. Reality

0368-4474, Winter 2015-2016

Lecture 10: Trusted Platform Architecture and SGX

Lecturer:

Eran Tromer

Guest lecturer:

Alon Jackson

Trusted Computing Architecture

Using presentation material from Dan Boneh, Stanford.

Background

- TCG consortium. Founded in 1999 as TCPA.
  - Main players (promoters): (>200 members)

    AMD, HP, IBM, Infineon, Intel,

    Lenovo, Microsoft, Sun

- Goals:
  - **Hardware protected (encrypted) storage**:
    - Only "authorized" software can decrypt data
    - e.g.: protecting key for decrypting file system
  - **Secure boot**: method to "authorize" software
  - **Attestation**: Prove to remote server what software is running on my machine.

Secure boot

History of BIOS/EFI malware:

- CIH (1998): CIH virus corrupts system BIOS
- Heasman (2007):
  - System Management Mode (SMM) “rootkit” via EFI
- Sacco, Ortega (2009): infect BIOS LZH decompressor
  - CoreBOOT: generic BIOS flashing tool

Main point: BIOS runs **before** any defenses (e.g. antivirus)

Proposed defense: lock system configuration (BIOS + OS)

Today: TCG approach

TCG: changes to PC

- Extra hardware: **TPM**
  - Trusted Platform Module (TPM) chip
    - Single 33MhZ clock.
    - Vendors: Atmel, Infineon, National, STMicro, ...
    - Cost: <$0.3
  - Integrated into other chips
    - Ethernet controller (Broadcom)
    - CPU's chipset (Intel)
- Software changes:
  - BIOS, EFI (UEFI)
  - OS and Apps

TPMs in the real world

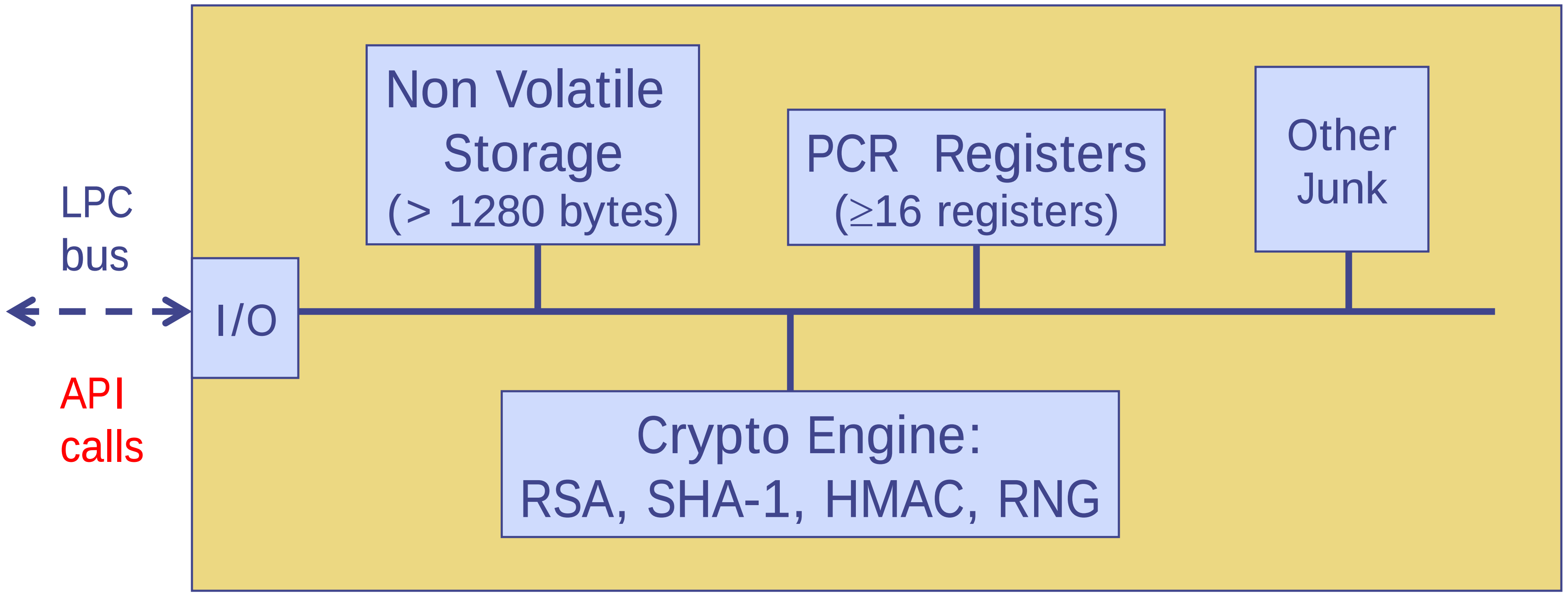
- TPMs widely available on laptops, desktops and some servers

- Software using TPMs:
  - File/disk encryption: BitLocker, IBM, HP, Softex
  - Attestation for enterprise login: Cognizance, Wave
  - Client-side single sign on: IBM, Utimaco, Wave

TPM Basics

- What the TPM does
- How to use it

Components on TPM chip



RSA:	1024, 2048 bit modulus
SHA-1:	Outputs 20 byte digest

Non-volatile storage

1. Endorsement Key (EK) (2048-bit RSA)
   - Created at manufacturing time. Cannot be changed.
   - Used for "attestation" (described later)

2. Storage Root Key (SRK) (2048-bit RSA)
   - Used for implementing encrypted storage
   - Created after running

     TPM_TakeOwnership(OwnerPassword, …)
   - Can be cleared later with TPM_ForceClear from BIOS

3. OwnerPassword (160 bits) and persistent flags

Private EK, SRK, and OwnerPwd never leave the TPM

PCR: the heart of the matter

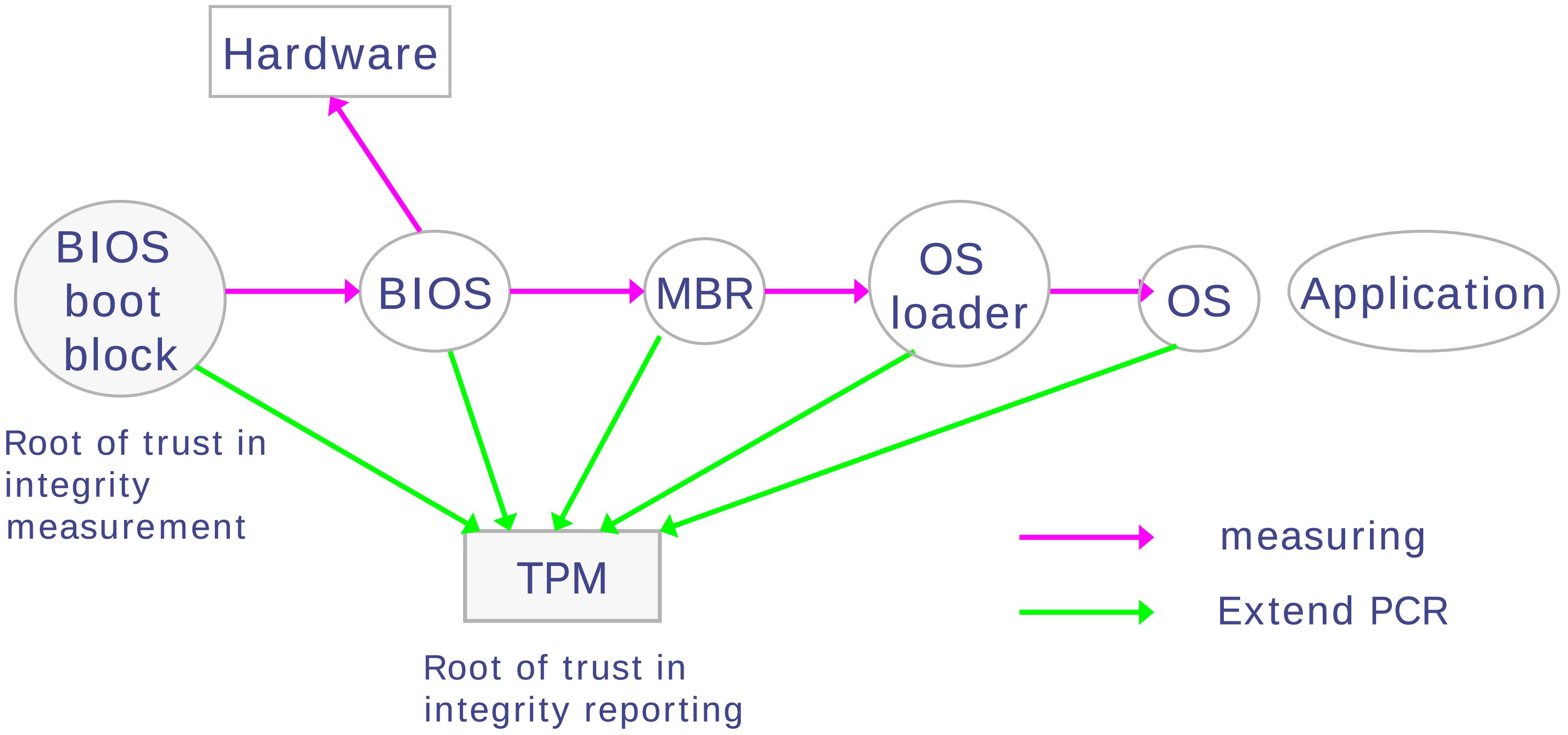
- *PCR: Platform Configuration Registers*
  - Lots of PCR registers on chip (at least 16)
  - Register contents: 20-byte SHA-1 digest (+junk)
- Updating PCR #n :
  - TPM_Extend(n,D): PCR[n] $\leftarrow$ SHA-1 (PCR[n] || D)
  - TPM_PcrRead(n): returns value(PCR(n))
- PCRs initialized to default value (e.g. 0) at boot time
  - TPM can be told to restore PCR values in NVRAM via
    TPM_SaveState and TPM_Startup(ST_STATE)
    for system suspend/resume

Using PCRs: the TCG boot process

- BIOS **boot block** executes
  - Calls TPM_Startup (ST_CLEAR) to initialize PCRs to 0
  - Calls PCR_Extend(n, <BIOS code>)
  - Then loads and runs BIOS post boot code
- BIOS executes:
  - Calls PCR_Extend(n, <MBR code>)
  - Then runs MBR (master boot record), e.g. GRUB.
- MBR executes:
  - Calls PCR_Extend(n, <OS loader code, config>)
  - Then runs OS loader

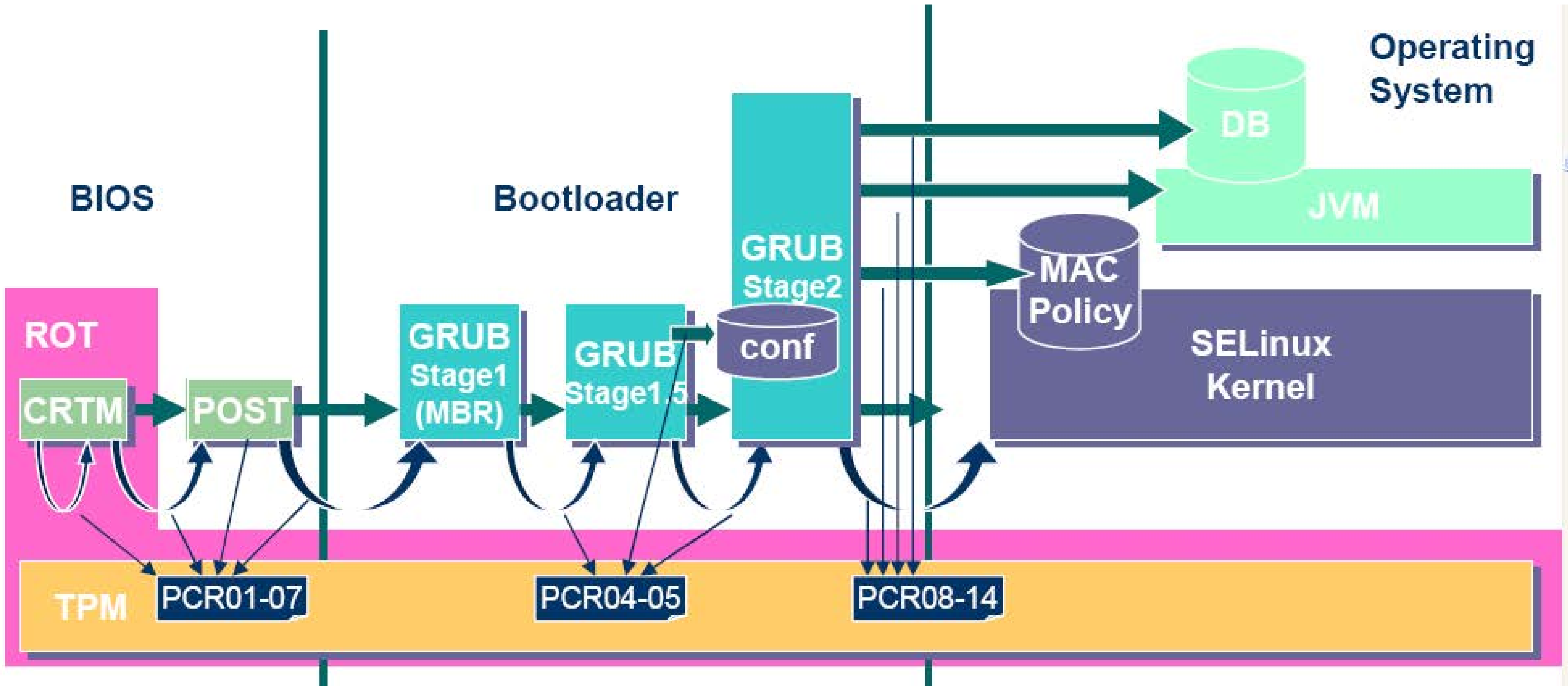
… and so on

In a diagram


Hardware
BIOS boot block
BIOS
MBR
OS loader
OS
Application
Root of trust in integrity measurement
TPM
Root of trust in integrity reporting
measuring
Extend PCR

- After boot, PCRs contain hash chain of booted software
- Collision resistance of SHA-1 ensures commitment

Example: Trusted GRUB (IBM' 05)



What PCR # to use and what to measure specified in GRUB config file

Using PCR values after boot

- Application 1: encrypted (a.k.a sealed) storage.

- Step 1: TPM_TakeOwnership(OwnerPassword, ...)
  - Creates 2048-bit RSA Storage Root Key (SRK) on TPM
  - Cannot run TPM_TakeOwnership again without OwnerPwd:
    - Ownership Enabled Flag $\leftarrow$ False
  - Done once by IT department or laptop owner.

- (optional) Step 2: TPM_CreateWrapKey / TPM_LoadKey
  - Create more RSA keys on TPM protected by SRK
  - Each key identified by 32-bit keyhandle

Protected Storage

- Main Step: Encrypt data using RSA key on TPM
  - TPM_Seal (some) Arguments:
    - keyhandle: which TPM key to encrypt with
    - KeyAuth: Password for using key `keyhandle’
    - PcrValues: PCRs to embed in encrypted blob
    - data block: at most 256 bytes (2048 bits)
      - Used to encrypt symmetric key (e.g. AES)
  - Returns encrypted blob.
- **Main point**: blob can only be decrypted with TPM_Unseal when PCR-reg-vals = PCR-vals in blob.
  - TPM_Unseal will fail othrewise

Protected Storage

- Embedding PCR values in blob ensures that only certain apps on certain platform configuration can decrypt data.
  - e.g.:   Messing with MBR or OS kernel will change PCR values.

Sealed storage: applications

- Lock software on machine:
  - OS and apps sealed with MBR’s PCR.
  - Any changes to MBR (to load other OS) will prevent locked software from loading.
  - Prevents tampering and reverse engineering

- Web server: seal server’s SSL private key
  - Goal: only unmodified Apache can access SSL key
  - Problem: updates to Apache or Apache config

- General problem with software upgrades/patches: Upgrade process must re-seal all blobs with new PCRs

Security?

- Resetting TPM after boot
  - Attacker can disable TPM until after boot, then extend PCRs arbitrarily
    (one-byte change to boot block) [Kauer 07]
  - Software attack: send TPM_Init on LPC bus allows calling TPM_Startup again (to reset PCRs)
  - Simple hardware attack: use a wire to connect TPM reset pin to ground
  - Once PCRs are reset, they can be extended to reflect a fake configuration.
- **Rollback attack** on encrypted blobs
  - e.g. undo security patches without being noticed.
  - Can be mitigated using Data Integrity Regs (DIR)
    - Need OwnerPassword to write DIR

Attestation

- **Goal**: prove to remote party what software is running on my machine.

- Good applications:
  - Bank allows money transfer only if customer’s machine runs “up-to-date” OS patches.
  - Enterprise allows laptop to connect to its network only if laptop runs “authorized” software
  - Quake players can join a Quake network only if their Quake client is unmodified.

- DRM:
  - MusicStore sells content for authorized players only.

Attestation: how it works

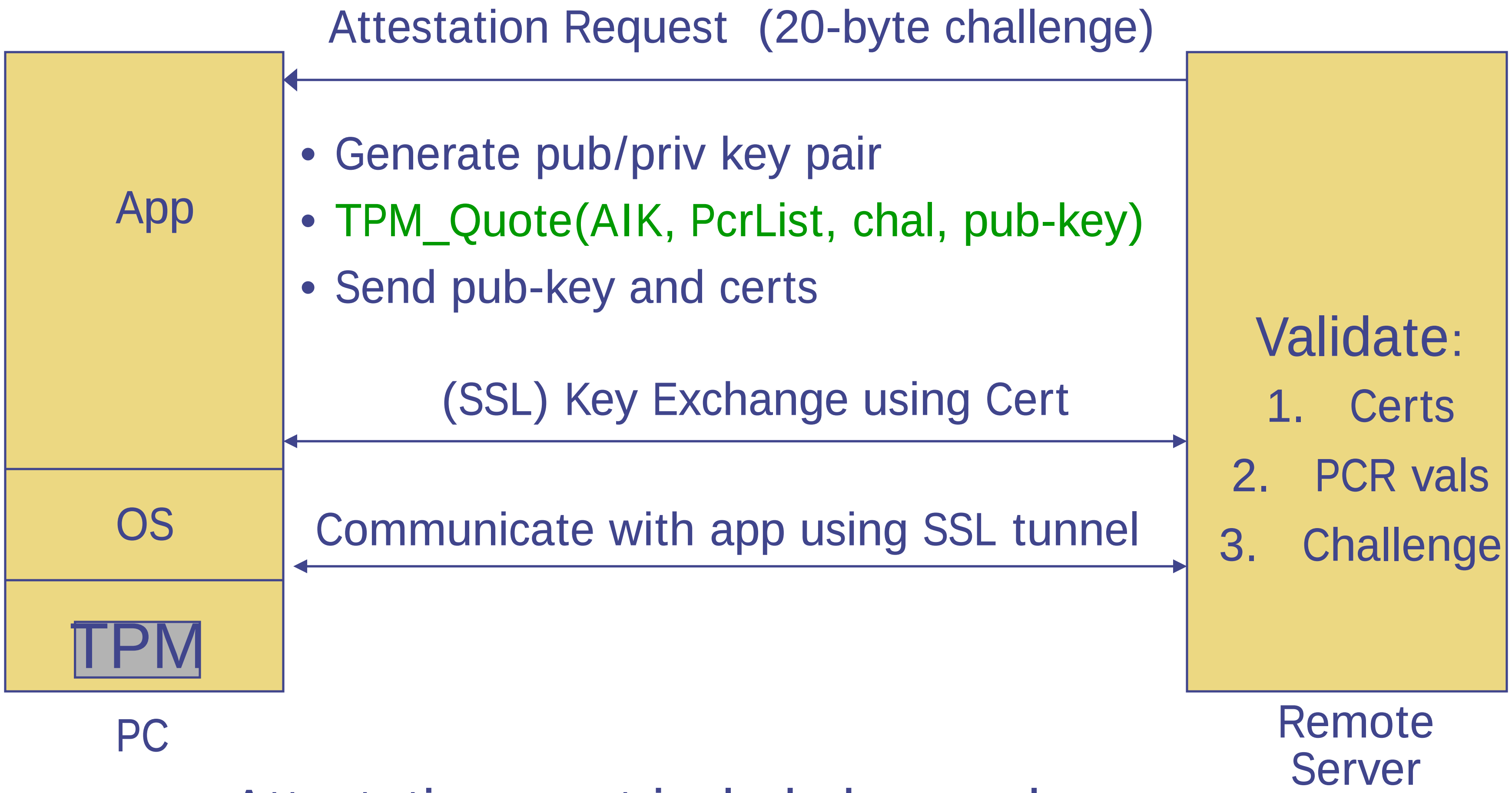
- Recall: EK private key on TPM.
    - Cert for EK public-key issued by TPM vendor.

- Step 1: Create Attestation Identity Key (AIK)
    - Involves interaction with a trusted remote issuer to verify EK
    - Generated:
AIK private+public keys, and a certificate signed by issuer

Attestation: how it works

- Step 2: sign PCR values (after boot)
  - Call TPM_Quote (some) Arguments:
    - keyhandle: which AIK key to sign with
    - KeyAuth: Password for using key `keyhandle’
    - PCR List: Which PCRs to sign.
    - Challenge: 20-byte challenge from remote server
      - Prevents replay of old signatures.
    - Userdata: additional data to include in sig.
  - Returns signed data and signature.

Using attestation (to establish an SSL tunnel)


Attestation Request (20-byte challenge)
App
OS
TPM
PC
Remote Server
(SSL) Key Exchange using Cert
Communicate with app using SSL tunnel

- Generate pub/priv key pair
- TPM_Quote(AIK, PcrList, chal, pub-key)
- Send pub-key and certs

Validate:
1. Certs
2. PCR vals
3. Challenge

- Attestation must include key-exchange
- App must be isolated from rest of system

Better root of trust: "late launch"

- **Late launch**: securely load OS/VMM, even on a potentially-compromised machine
- DRTM – Dynamic Root of Trust Measurement
- New CPU instruction:
  Intel TXT: **SENTER** AMD: **SKINIT**
- Atomically does:
  - Reset CPU. Reset PCR 17 to 0.
  - Load given Secure Loader (SL) code into I-cache
  - Extend PCR 17 with SL
  - Jump to SL
- BIOS boot loader is no longer root of trust
- Avoids TPM_Init attack: TPM_Init sets PCR 17 to -1

Trusting the CPU?

- Trust the vendor?
- System Management Mode (SMM)
  - Special execution mode with protected memory, “below” OS and hypervisor
  - Survives TXT launch, so compromised BIOS could load a malicious SMM that circumvents TXT.
- Intel Management Engine (ME)
  - A microcontroller embedded in the processor, “next to” the OS and hypervisor
  - Independent of OS security policy.
  - Applications: originally, remote administration. Nowadays: EPID (chip-specific keys), Boot Guard (checks signature on boot code), Protected Audio and Video Path, SGX (secure enclaves)
  - Trustworthy?
    - ME includes its own, potentially vulnerable, custom OS and apps.
    - No public documentation of implementation and logic.
    - Could be used as an ideal rootkit / trojan horse.

Security?

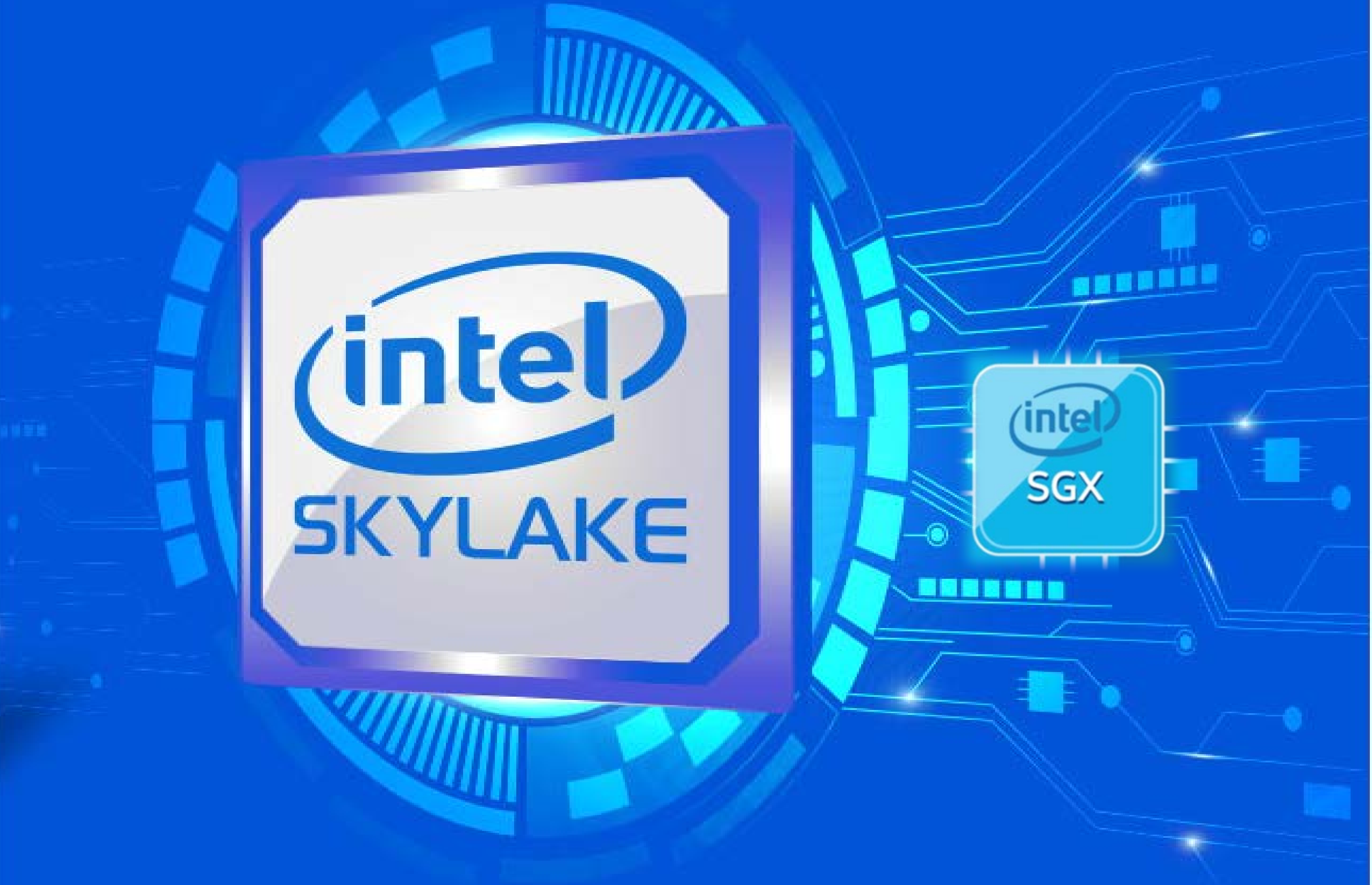
- **Resetting TPM after boot**
  - Attacker can disable TPM until after boot, then extend PCRs arbitrarily
    (one-byte change to boot block) [Kauer 07]
  - Software attack: send TPM_Init on LPC bus allows calling TPM_Startup again (to reset PCRs)
  - Simple hardware attack: use a wire to connect TPM reset pin to ground
  - Once PCRs are reset, they can be extended to reflect a fake configuration.
- **Rollback attack** on encrypted blobs
  - undo security patches
- **Large trusted base**
  - Long, complex and potentially vulnerable chain of trust: OEM's drivers, vendor's peripherals, OS's vendors.

Protecting code on an untrusted platform

- Can we run sensitive code on a potentially-compromised platform, without rebooting/replacing it?
  - Many ways to read and corrupt code!
- **Secure "enclave"** using CPU hardware
  - Possible with SENTER/SKINIT but cumbersome (*Flicker* project)
  - Intel Software Guard Extensions (SGX)
    (discussed later)
  - ARM TrustZone
- **Cryptography**
  - Fully-homomorphic encryption
  - Succinct zero-knowledge proofs (SNARKs) and Proof-Carrying Data

SGX

Using presentation material from Intel.



Some slides are courtesy of intel©

Why SGX?

Software security on commercial CPUs is pursued for many years

- SW solutions have limited potential.
- Closed HW systems have limited functionality.
- Traditional TPMs have large TCB – commonly measure all platform.

TXT overcame some of these by introducing a Dynamic Root of Trust:

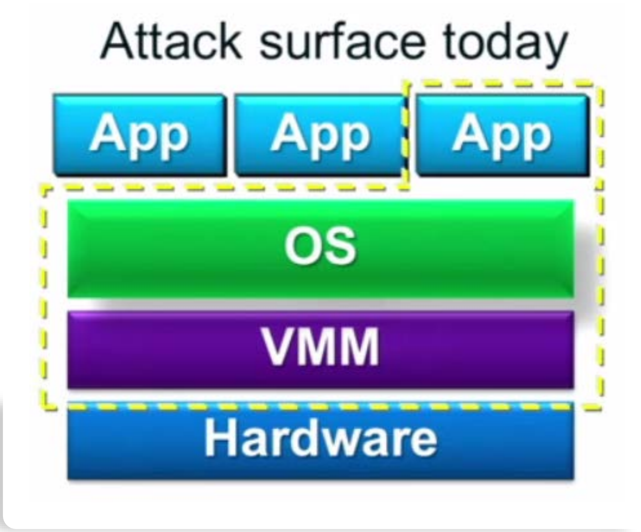
✓ Dedicated TCB “mini OS”.

✓ Run only sensitive logic in trusted OS.

✓ Thus, needs to measure only mini OS and sensitive logic app.

But TXT still:

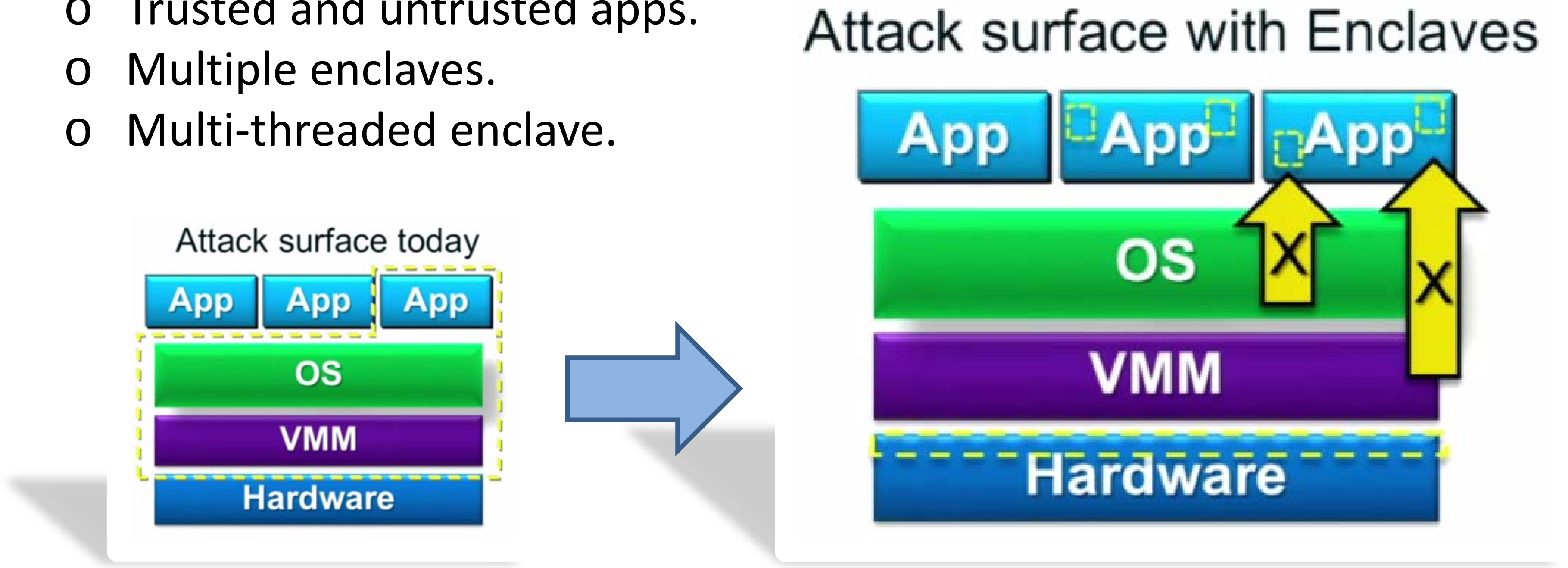
- Halts main OS, restarts mini-OS and clears CPU.
- Doesn’t support continuous run of trusted and untrusted apps.
- Cumbersome to develop for.
- Large TCB - lots of potentially vulnerable SW. Numerous vulnerabilities - Most Hazardous: Compromised BIOS can load a malicious SMM that survives TXT launch.



Why SGX?

SGX introduces the ENCALVE:

- A protected TEE (Trusted Execution Environment) container.
- Shrinks TCB (Trusted Computing Base) to HW and sensitive app logic.
- Extends HW TCB on to enclaves in ring-3.
- Runs TEE under untrusted OS or VMM.
- Supports continuous run of:
  - Trusted and untrusted apps.
  - Multiple enclaves.
  - Multi-threaded enclave.

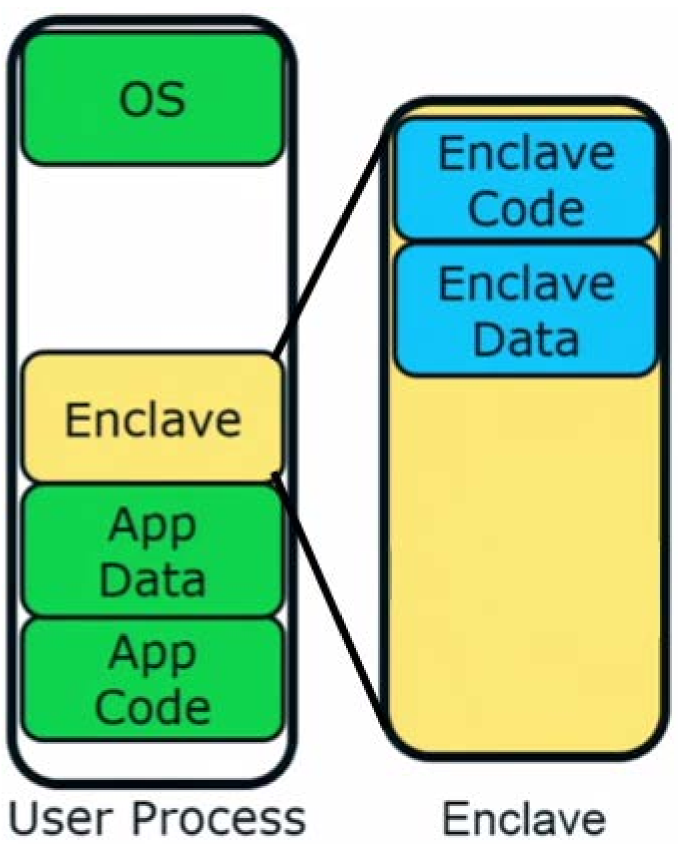


While maintaining application Integrity and confidentiality.

How?

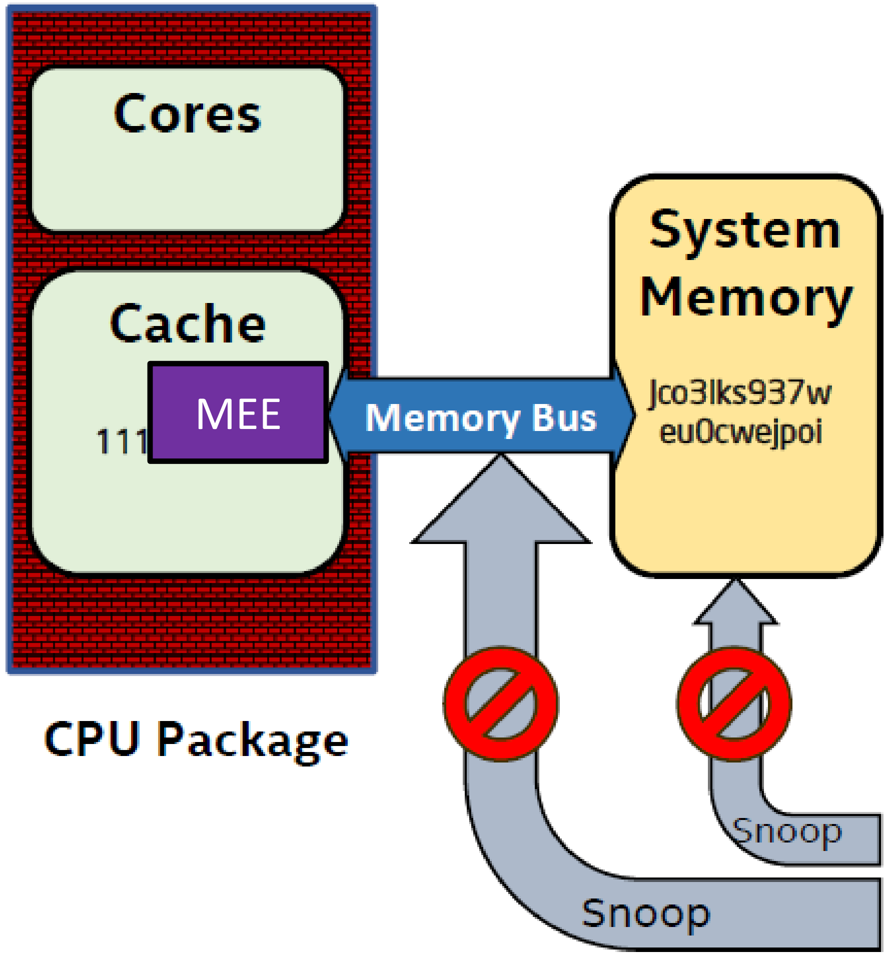
- Dedicated HW
- 18 new ISA instructions
- Internal FW data structures
- Tailored cryptographic functions
- Suitable protocols

The Enclave – A trusted execution environment embedded in a process



- Is part of the application & has full access to its memory.
- Measured and verified by HW on load.
- Operation visible only within CPU borders.
- Manageable by the OS, e.g:
  - CPU time slots
  - Paging and memory policy
- Secrets are provisioned or acquired only after enclave initialization.

Security Perimeter

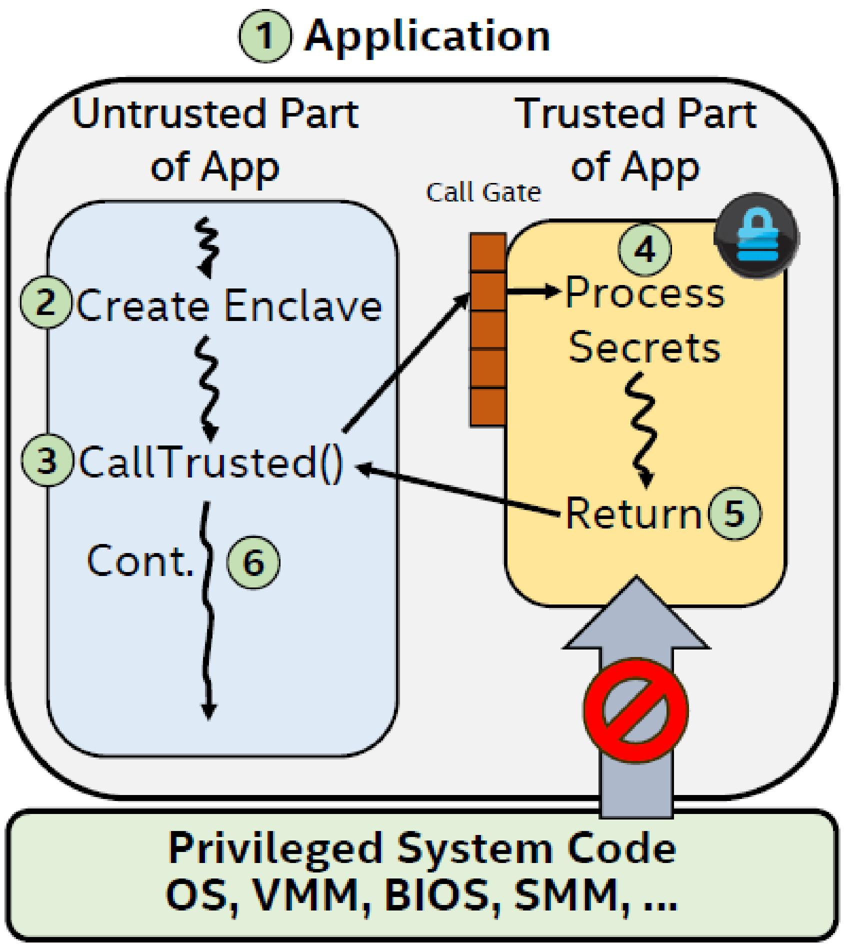


- Security perimeter is the CPU package boundary.
- Data and code unencrypted inside CPU package.
- Data and code outside CPU is encrypted and integrity checked.

- Single chip TCB avoids inter-chip HW attacks that threaten TPMs.
- If the single TCB is the CPU, then we gain the opportunity for richer semantics by understanding platform state & app code.

* **MME** – Memory Management Engine, elaborated shortly.

Execution Flow

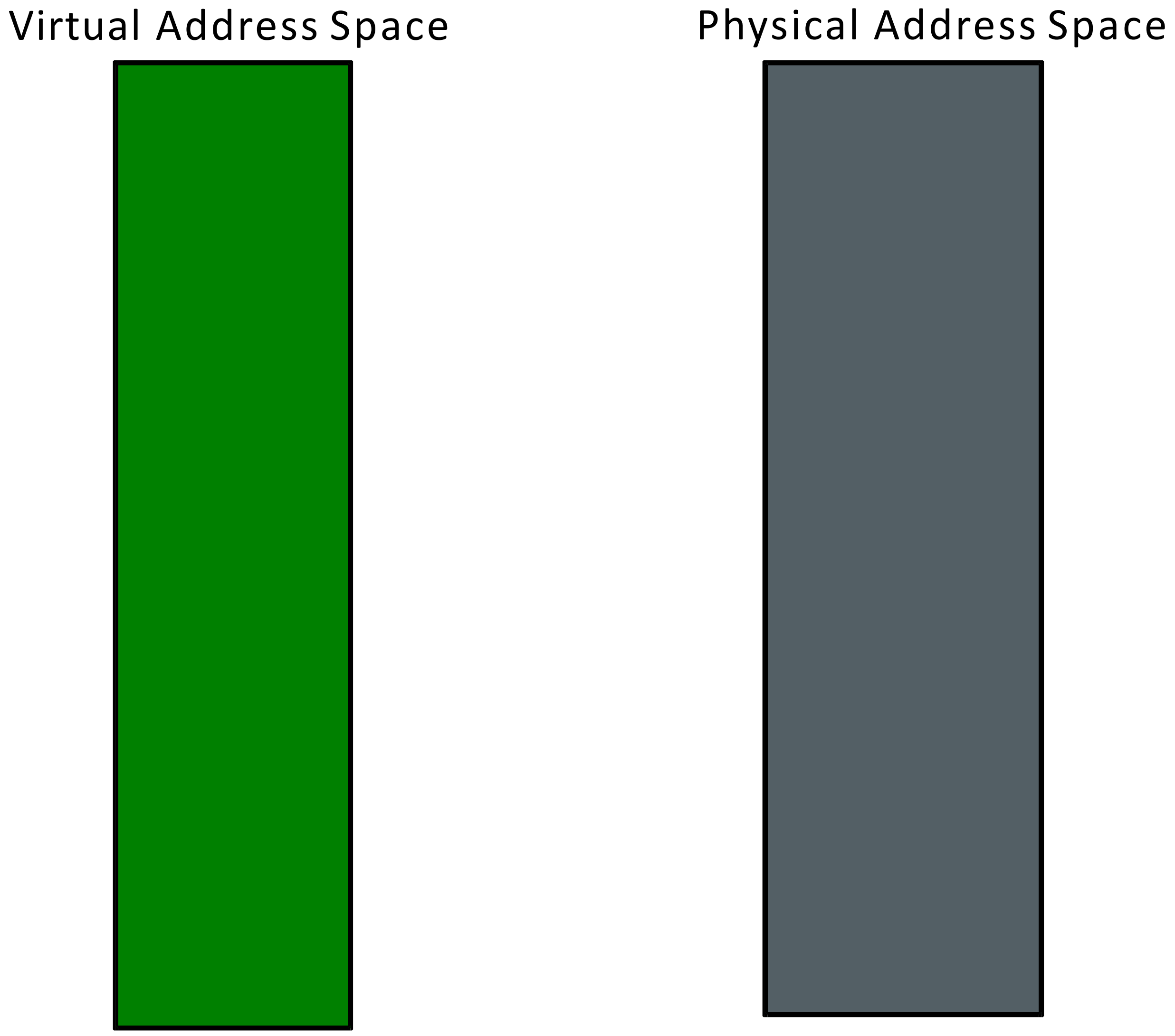


1. Enclavize sensitive App parts defines a partition to trusted and untrusted.
2. App runs & creates enclave in trusted memory.
3. Trusted function transitions flow to the enclave.
4. Enclave **sees all process data**, but **external access to enclave is denied**.
5. Trusted function returns.
6. App continues normal execution.

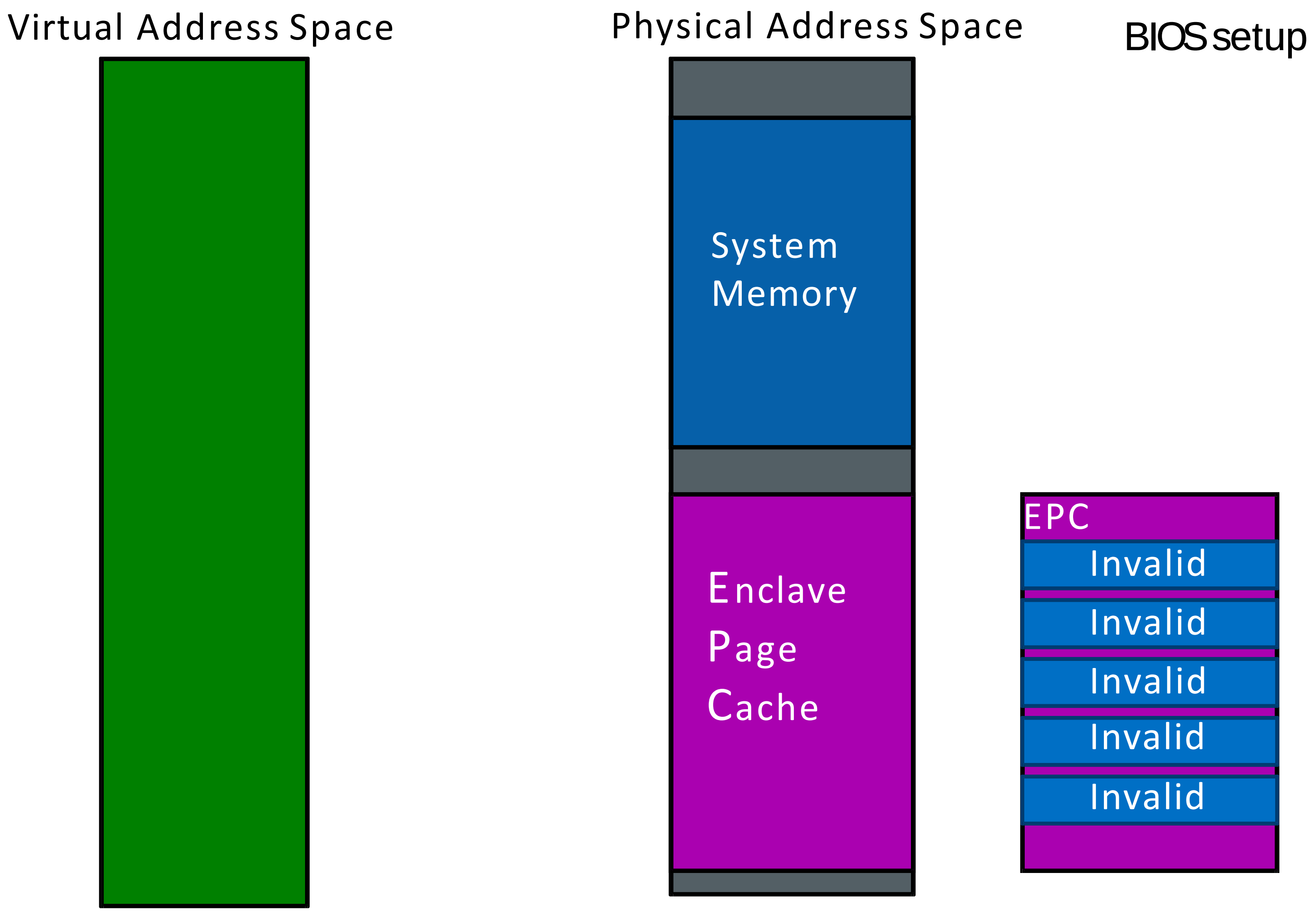
Motivation – Some Usage Examples:

- **Cloud** - Assure customers about the security standard of the “cloud enclave”. Confidential, also from vendors with physical access.
- **HSM** - Implement HSM’s functionality and security such that existing software that supports HSMs can be adapted to using SGX instead.
- **DRM** – On-line content provider that provides content only to authorized players hosting a trusted player enclave.
- **TOR** - A node that could prove to users that it is not backdoored by its own admin and does not keep a log of how connections were routed.

Enclave Life Cycle

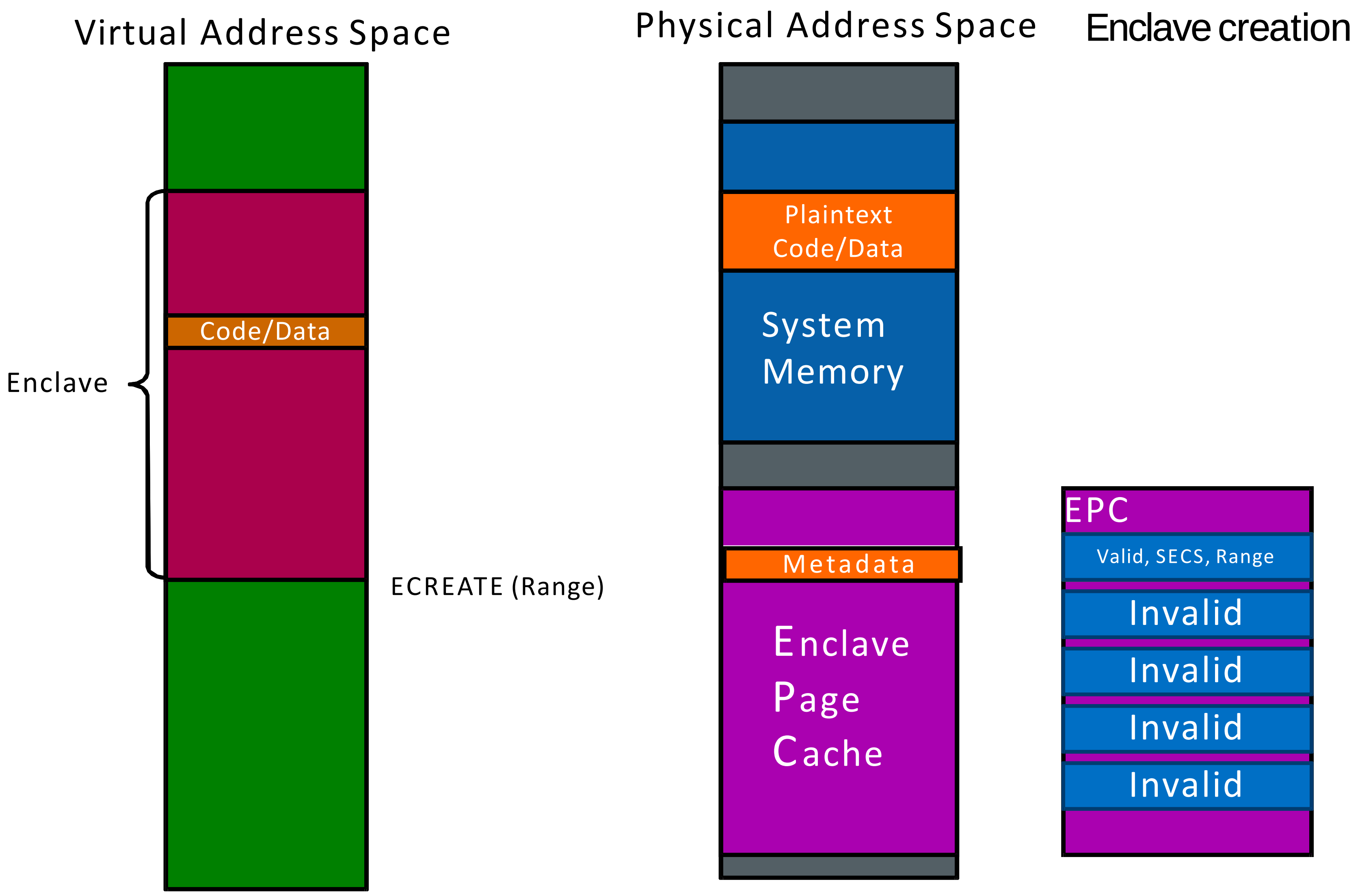


Enclave Life Cycle



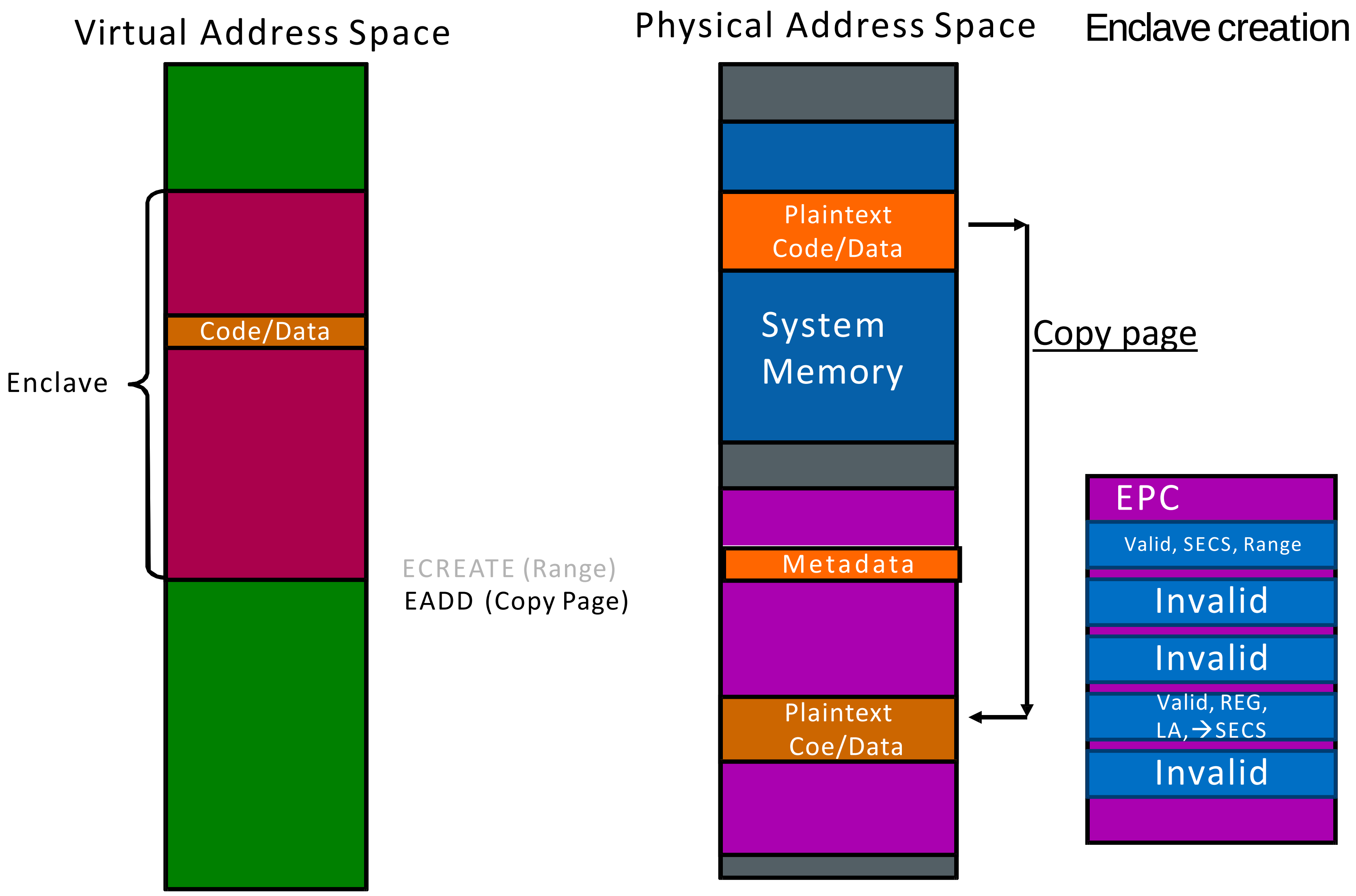
BIOS reserves memory address range for SGX use.

Enclave Life Cycle



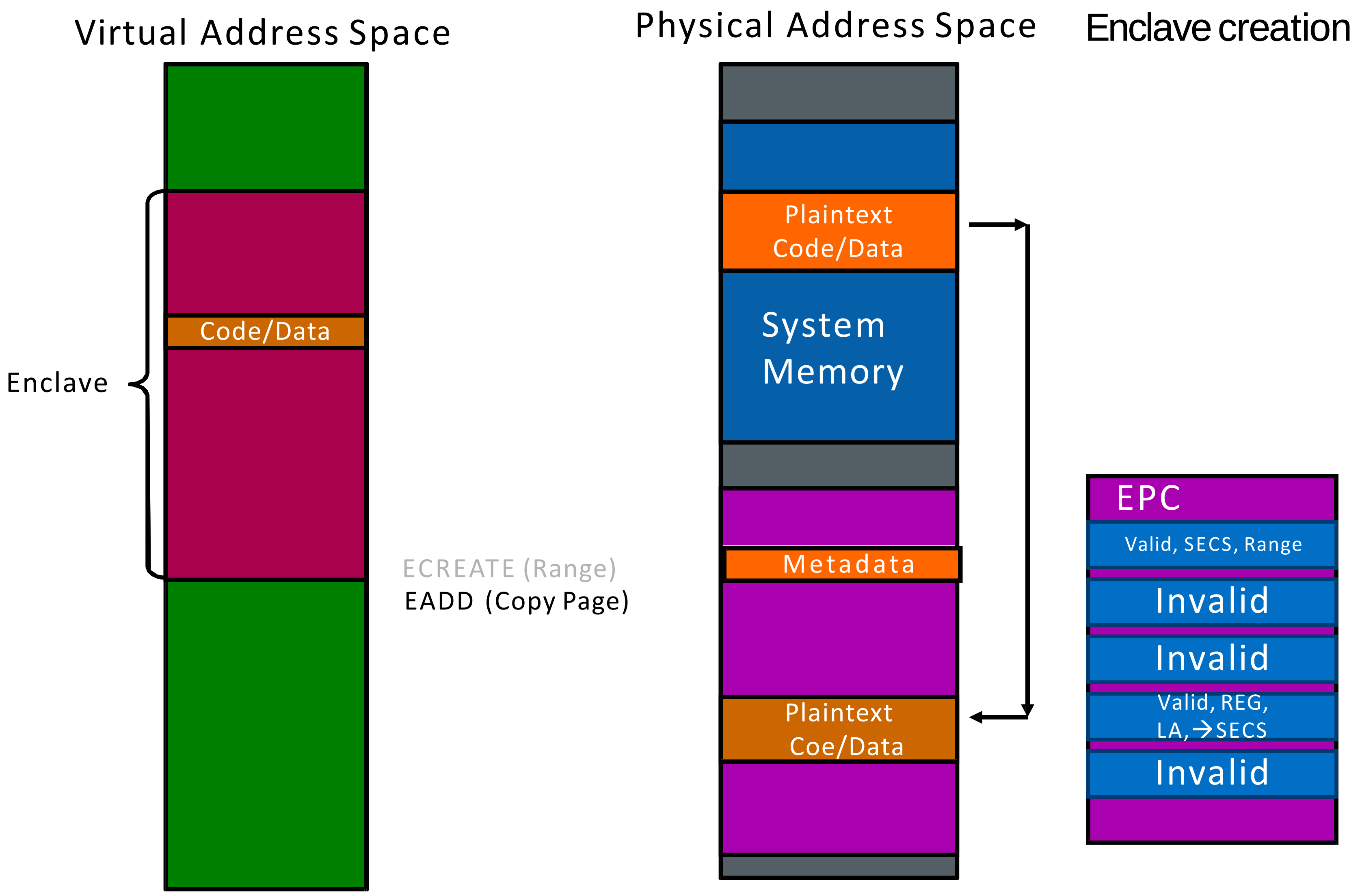
ECREATE - Stores enclave attributes (mode of operation, debug, etc.) in metadata.

Enclave Life Cycle



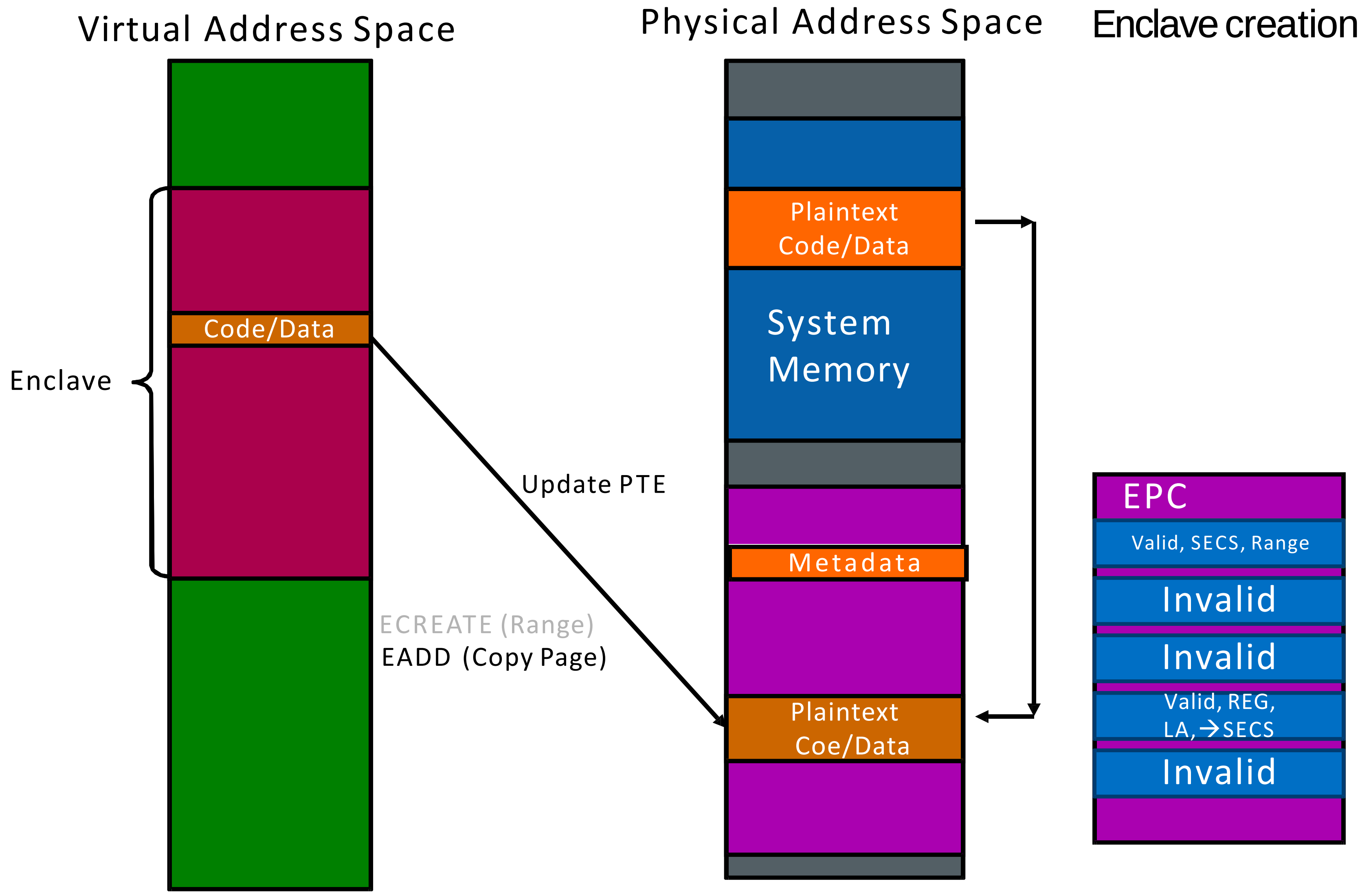
EADD – Commits new pages to enclave & updates security metadata.

Enclave Life Cycle

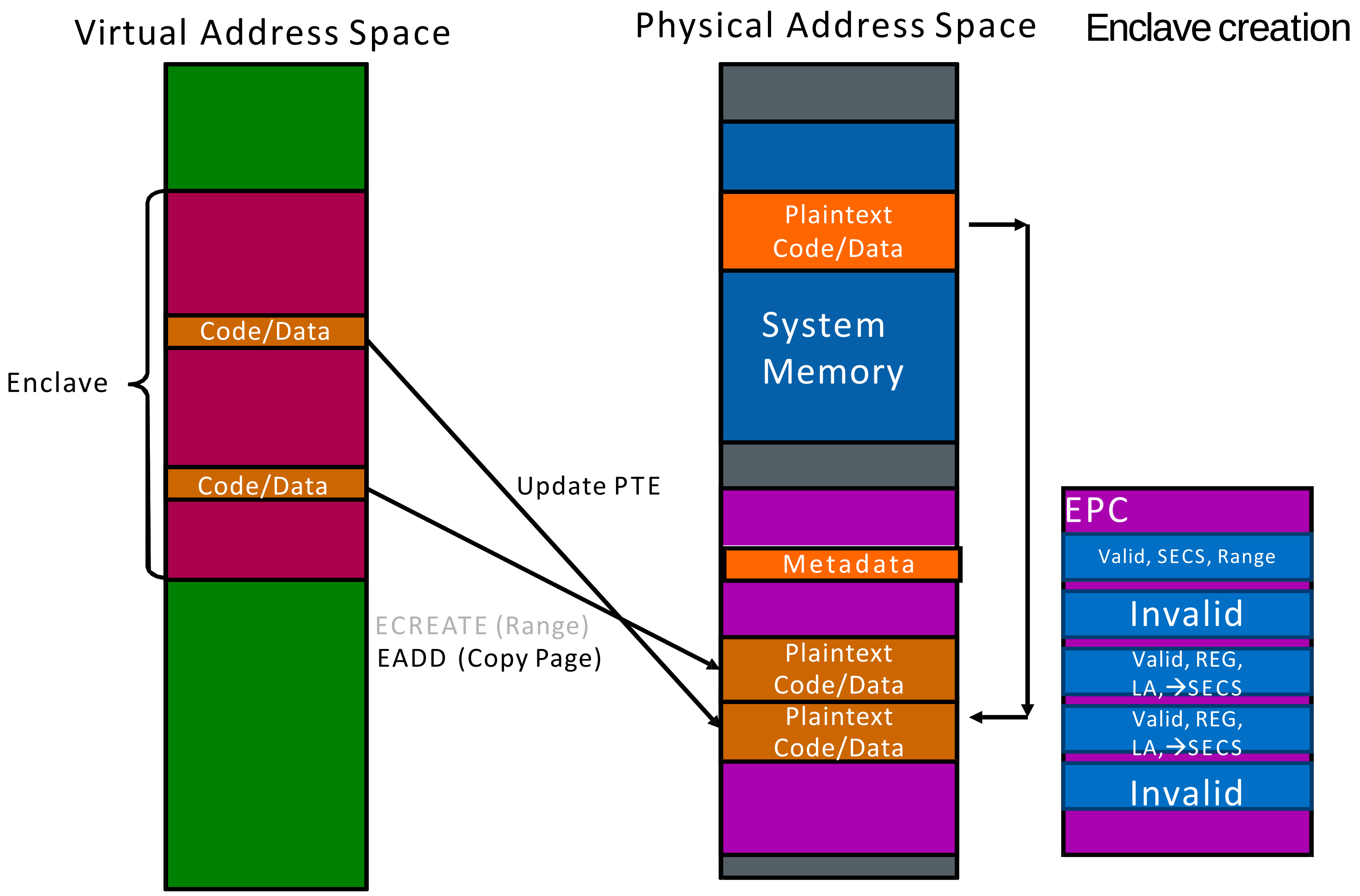


EADD – Commits code, data or SGX control structure page types.

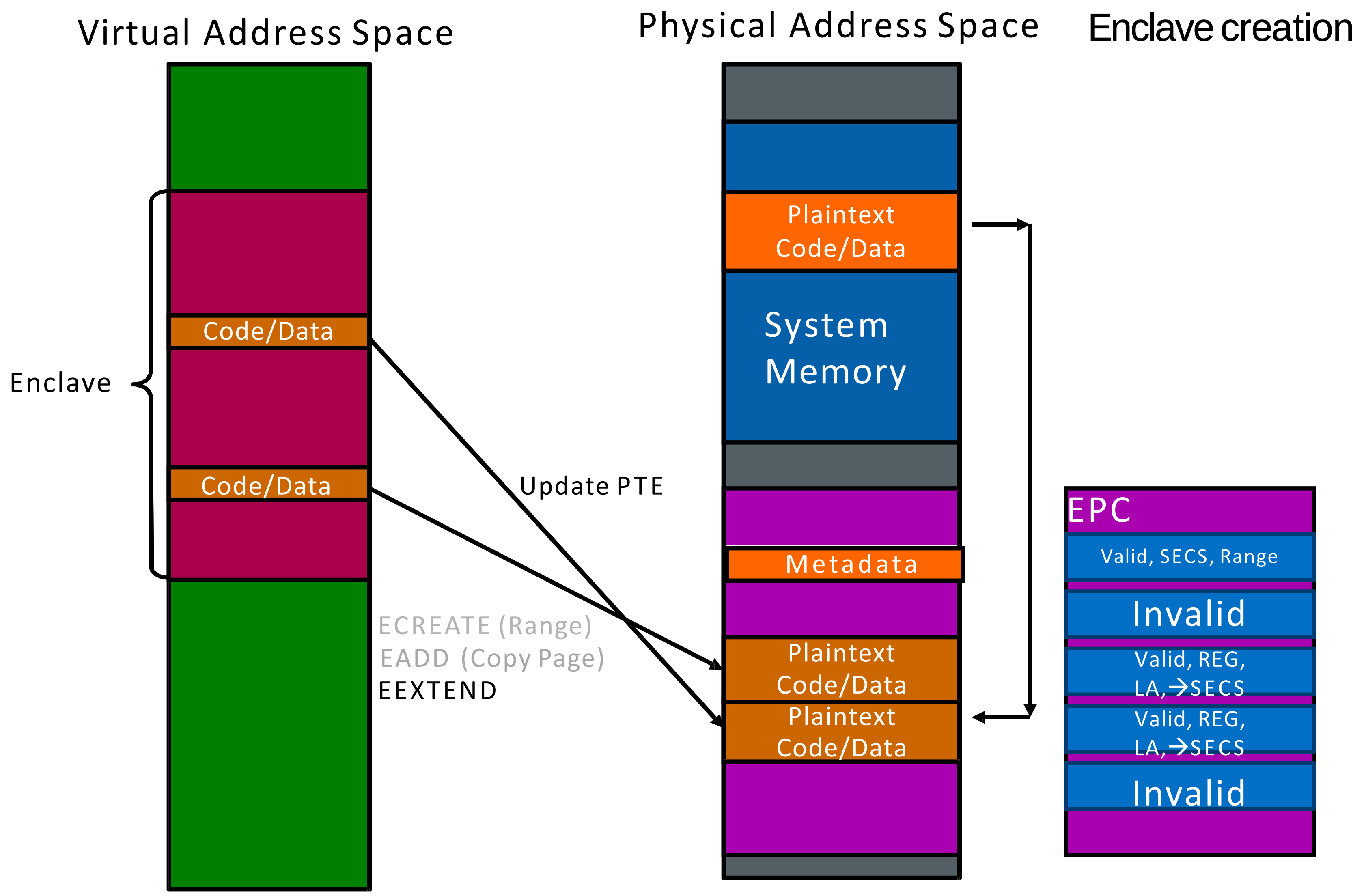
Enclave Life Cycle



Enclave Life Cycle

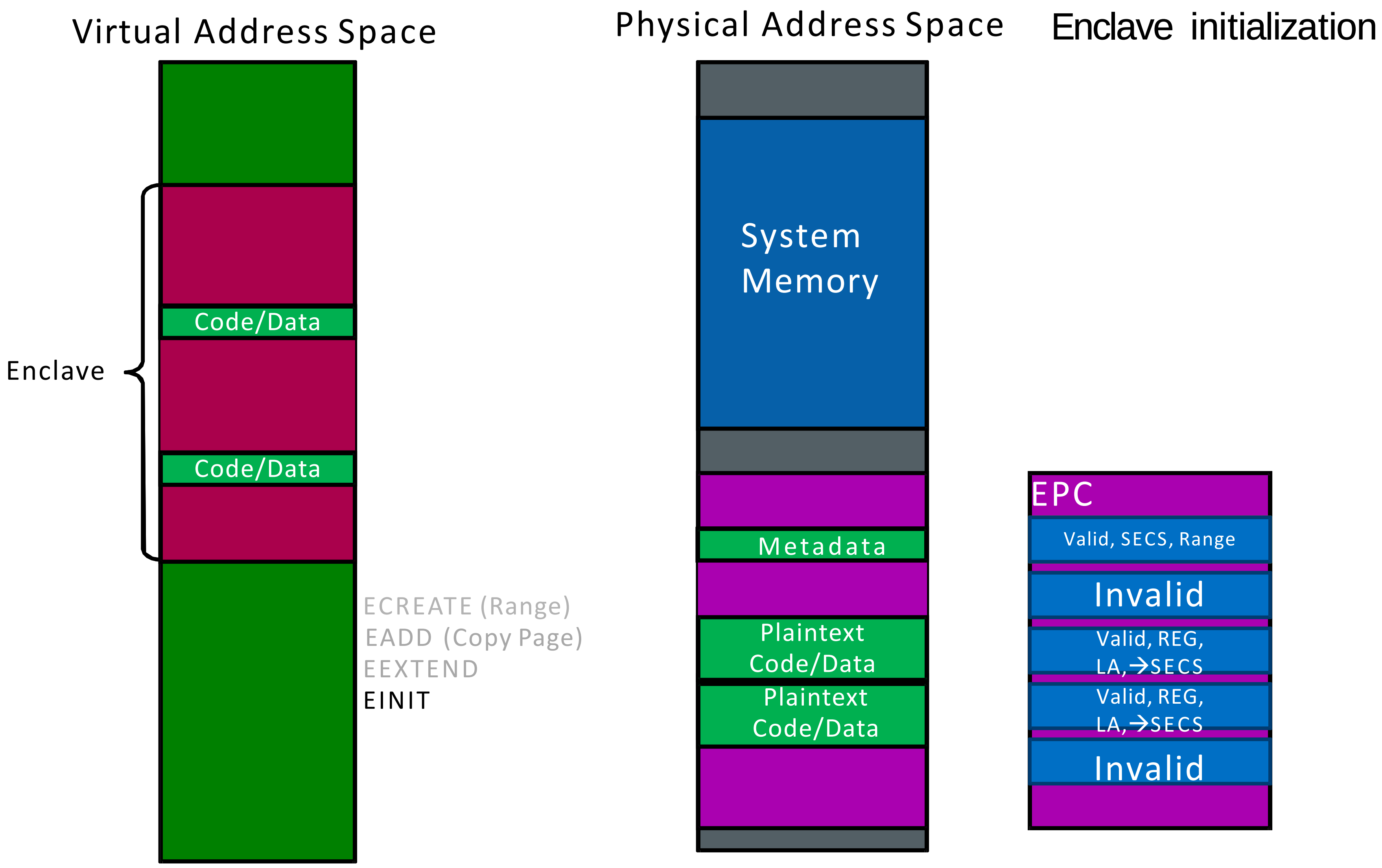


Enclave Life Cycle



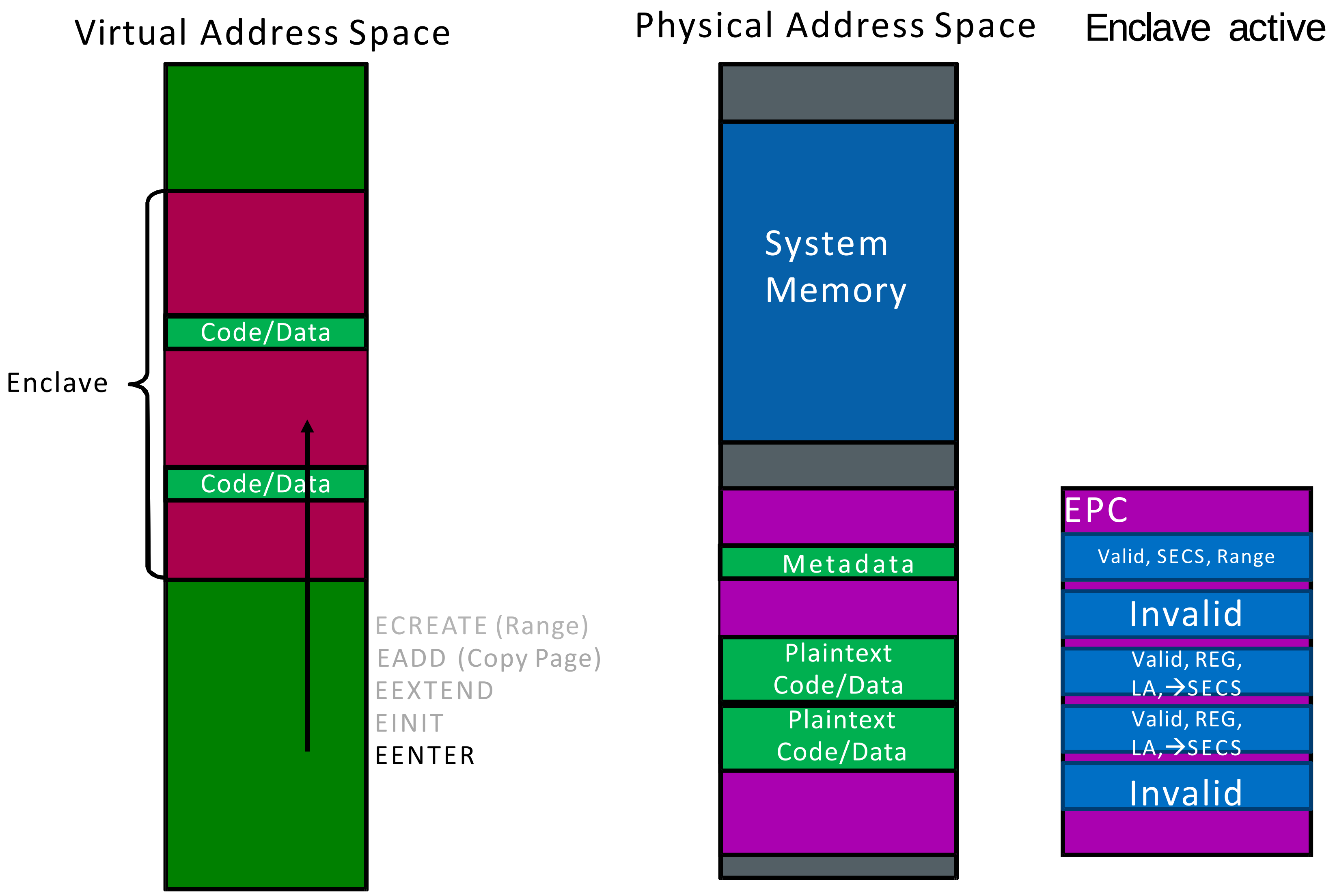
EEXTEND - Measures the enclave with SHA256. (detailed shortly)

Enclave Life Cycle



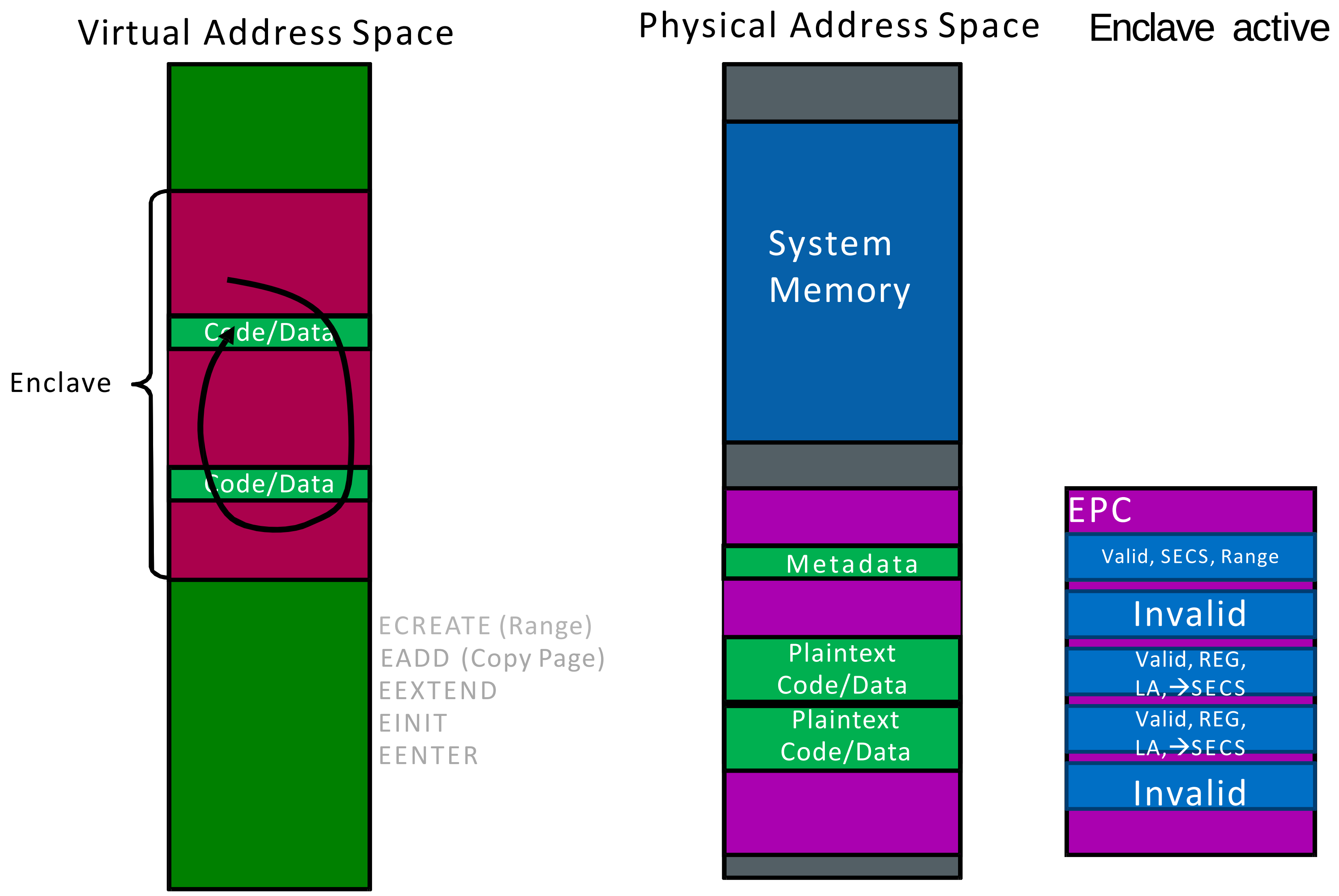
EINIT - Finalizes measurements, validates them & enables enclave's entry use.

Enclave Life Cycle



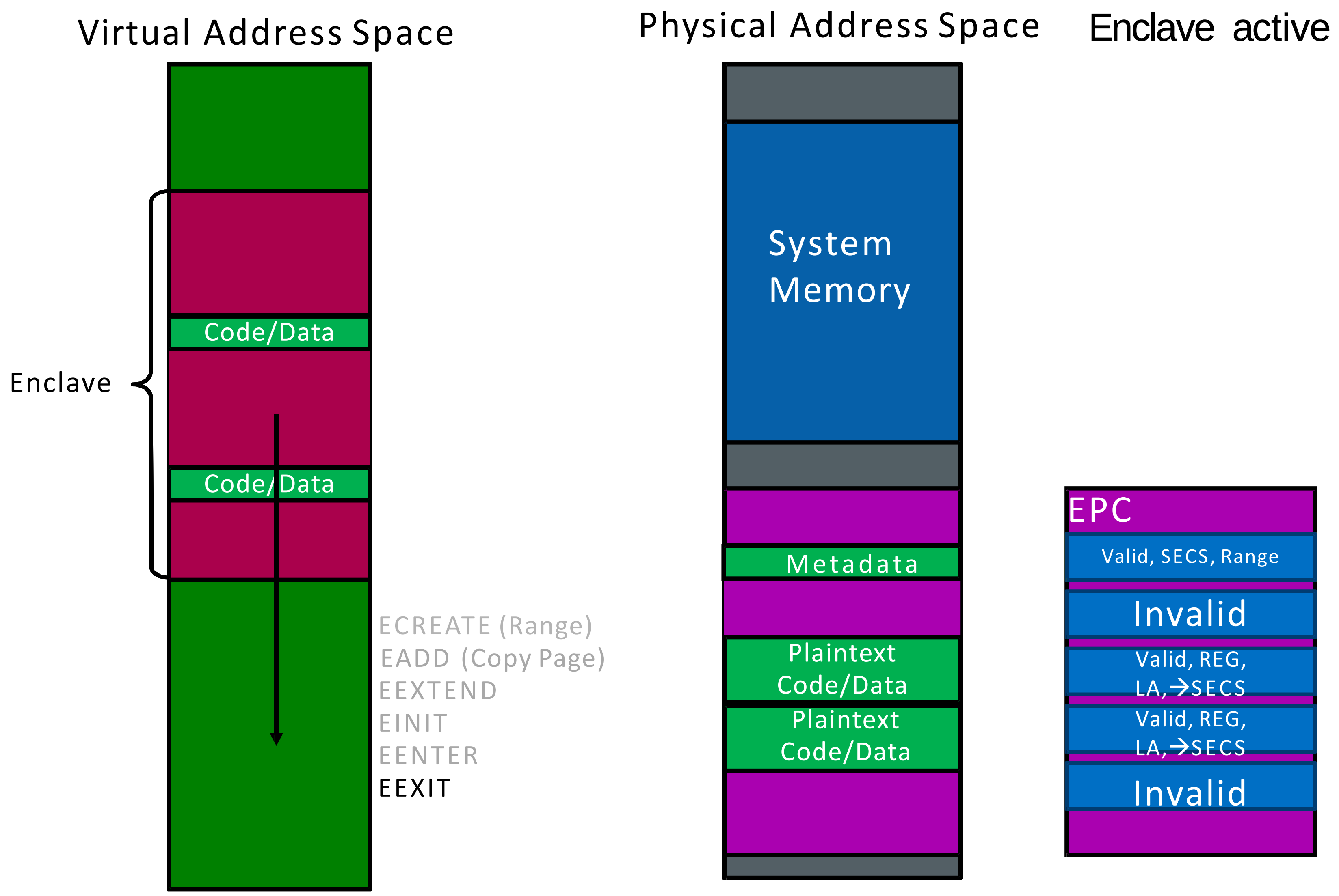
EENTER - Verifies enclave entry & sets CPU operation mode to "enclave mode".

Enclave Life Cycle



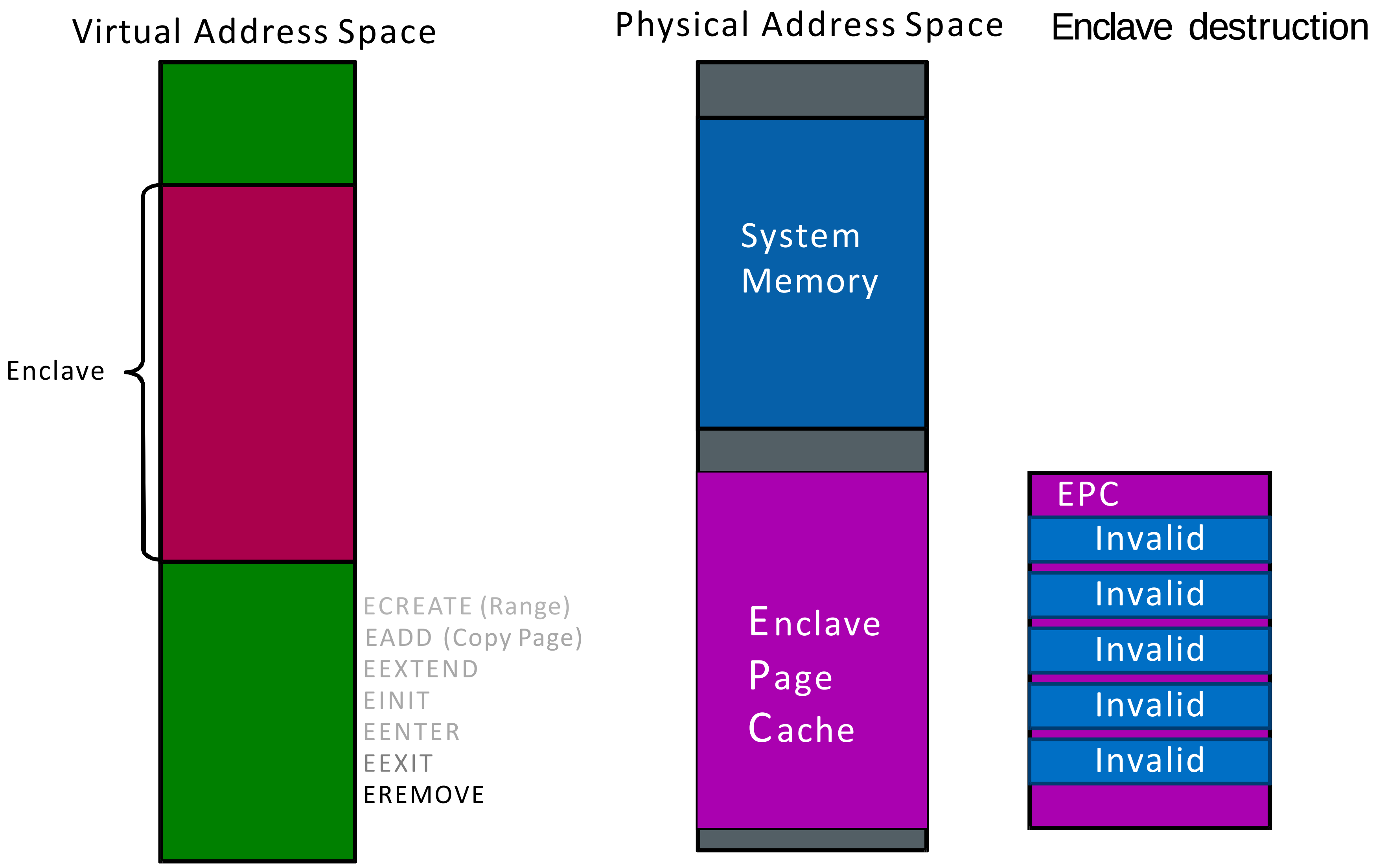
Enclave flow is executed using the secured address range, obscured from **all** other SW.

Enclave Life Cycle



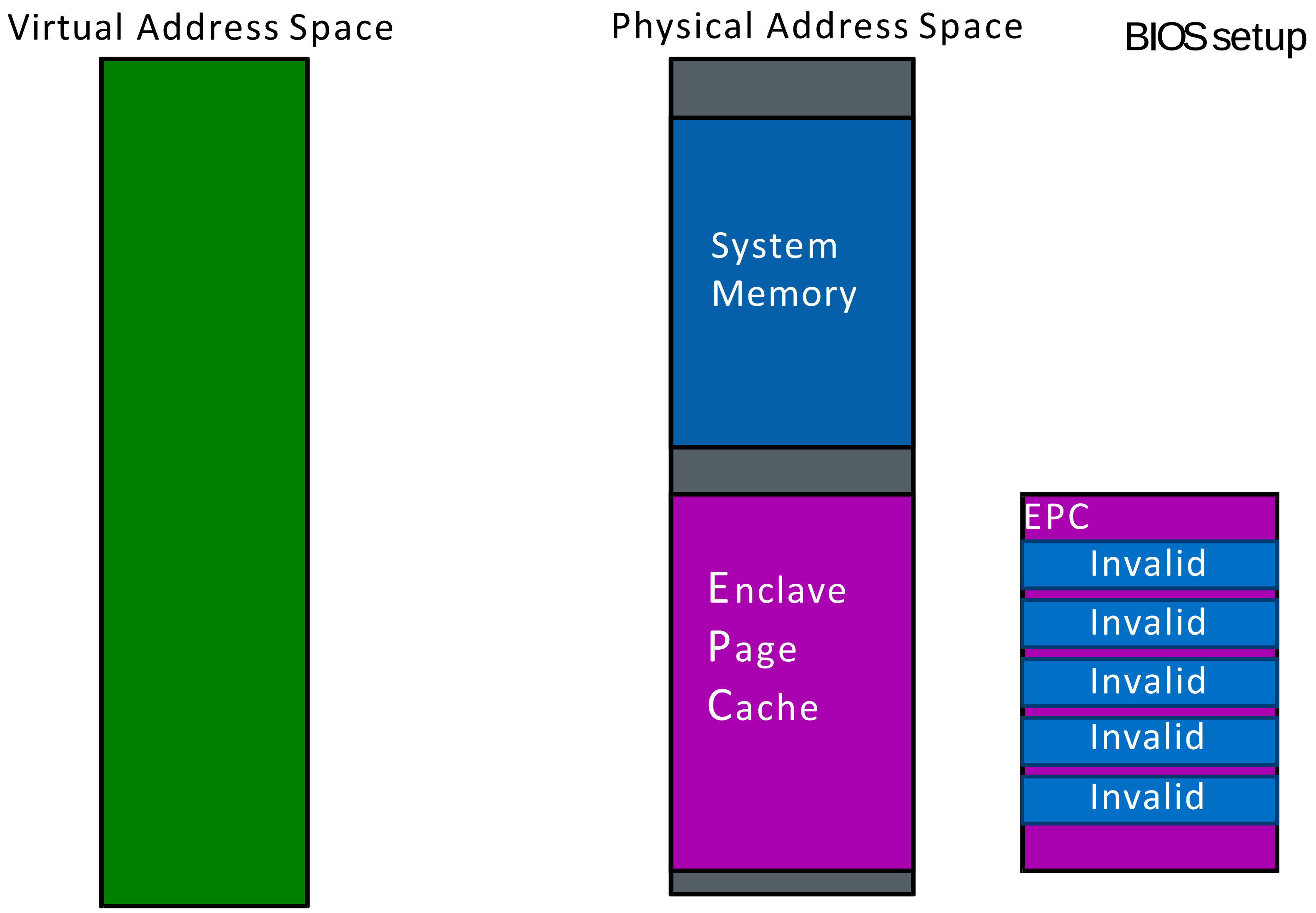
EEXIT – Clears CPU cache & Jumps out of enclave back to OS instruction address.

Enclave Life Cycle



EREMOVE – Clears enclave's trusted virtual address range reserved by ECREATE.

Enclave Life Cycle



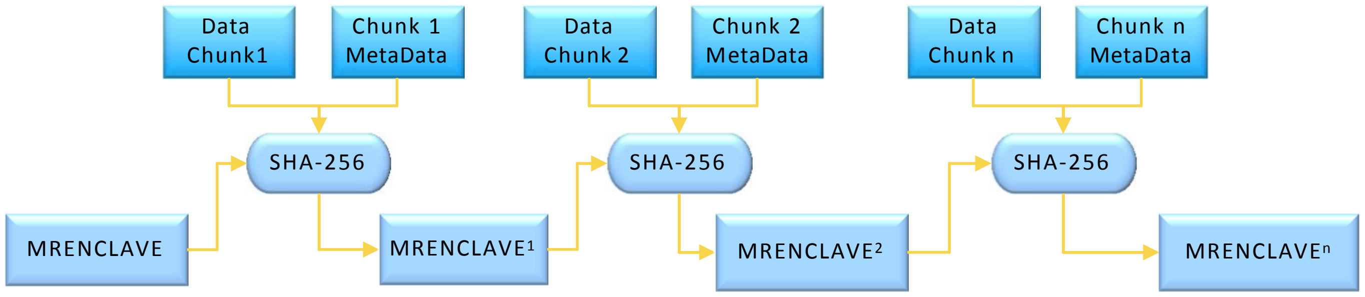
Enclave Measurement

When building an enclave, Intel® SGX generates a cryptographic log of all the build activities

- Content: Code, Data, Stack, Heap
- Location of each page within the enclave
- Security flags being used
- Order in which it was built

MRENCLAVE ("Enclave Identity") is a 256-bit digest of the log

- Represents the enclave's software TCB



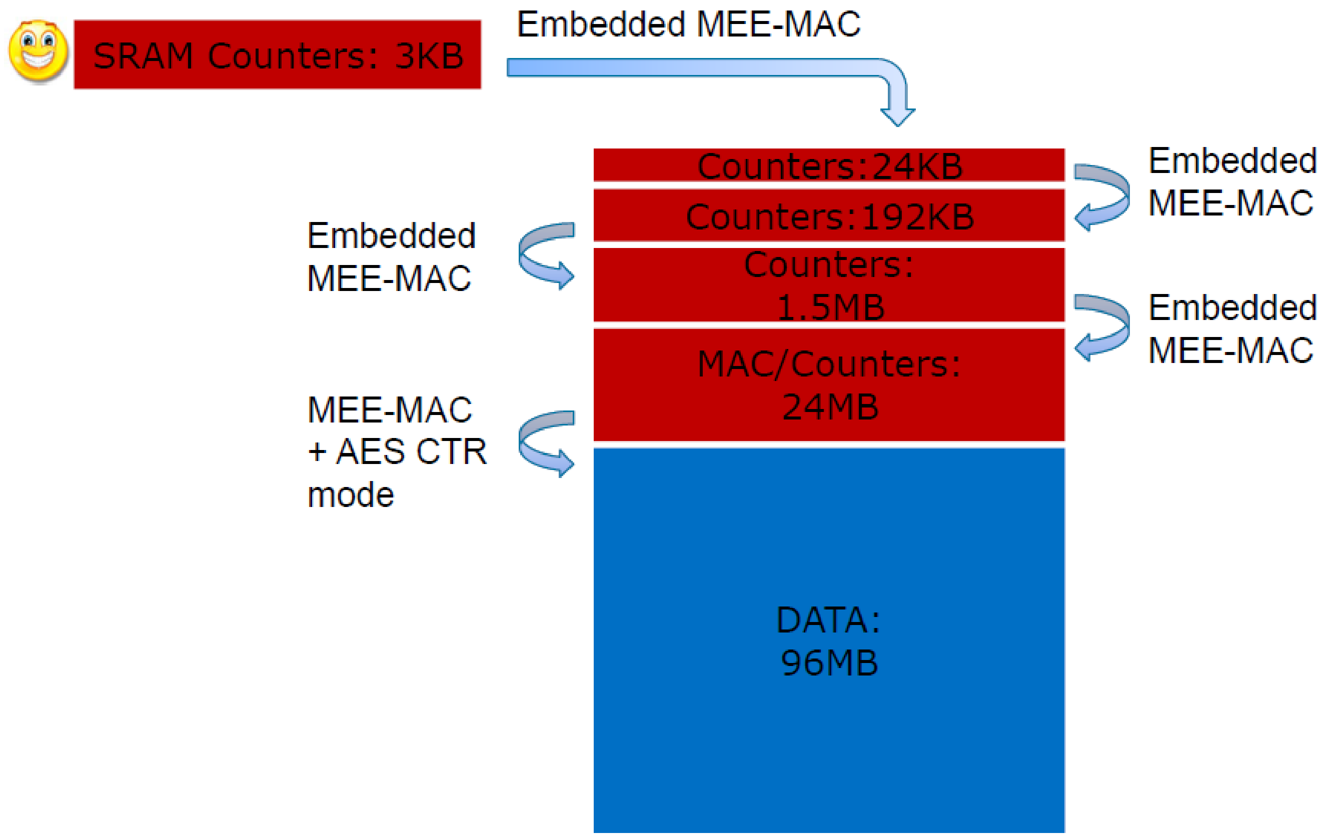
Memory Encryption Engine (MEE)

- Resides within the CPU as a Memory Controller extension.
- **Encrypts, decrypts page swaps and integrity checks them thus, providing data confidentiality and integrity.**
- TRNG generates at boot time separate keys for encryption and integrity.
- Keys are held in MEE registers, **accessible only to HW**.
- Does not hide the fact that data is written to the DRAM , when it is written, and to which physical address.
- Defends system memory from cold boot attacks.
- Uses version and counter controls to prevent page replay attack. **Problem – A lot of metadata stored in CPU…**

SGX Anti replay attack under HW limitations

Uses **version and counter controls** to prevent page replay attack.

- Keeping this metadata for every evicted page is not scalable in SRAM terms.
- How can we "compress" this?



- Can be implemented as a merkle-tree.

Halfway recap

- Enclave properties as a TEE
- SGX Security Perimeter
- lifecycle and execution flow
- Enclave Measurement
- Memory Encryption Engine

- **Sealing**
- **Attestation**
- **Provisioning**

- Overview summery
- SGX usages
- Q&A and open discussion

Sealing

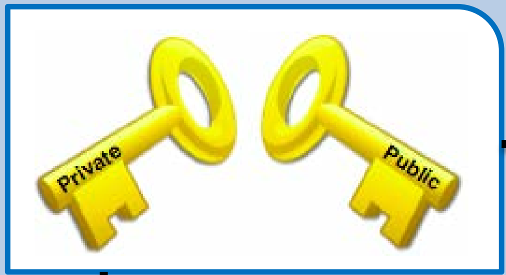
- Definition: Cryptographically protecting data when it is stored outside enclave.
- Sealing enables enclave to pass data between consecutive runs.
- Future enclave instantiation can acquire former provisioned sensitive data.

  (Enclave shouldn’t ship with sensitive data, but be provisioned with it after instantiation.)
- Newer SVN (security version number) enclave can read older SVN data using a Key Recovery Transformation.

Sealing Actors

Sealing Authority – Developer

Enclave ISV that Signs the enclave **certificate** with its RSA-3072 Private key.



SIGSTRUCT - Enclave certificate.

- Sealing Authority **Public key** (Used by HW to verify **Certificate**).
- Enclave identity aka **MRENCLAVE** (Used by HW to verify enclave Integrity).
- ISV SVN
- Other enclave attributes.

Sealing Authority signature

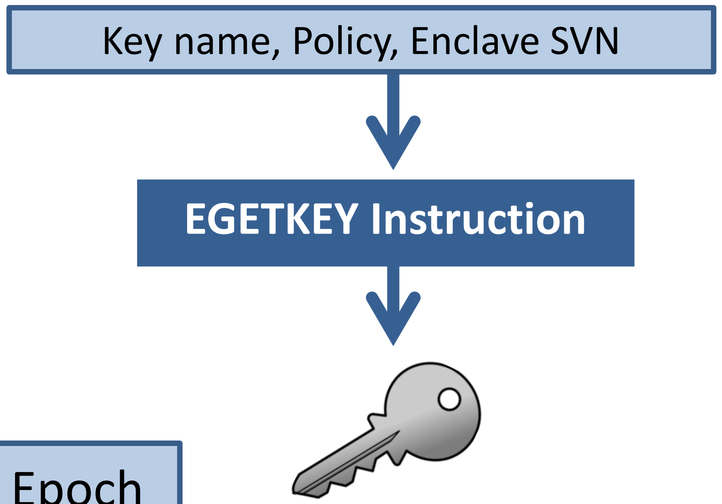
MRSIGNER - Sealing Identity.

- Created on enclave initialization.
- Accessible only by the TCB.
- **Used to seal enclave data.**

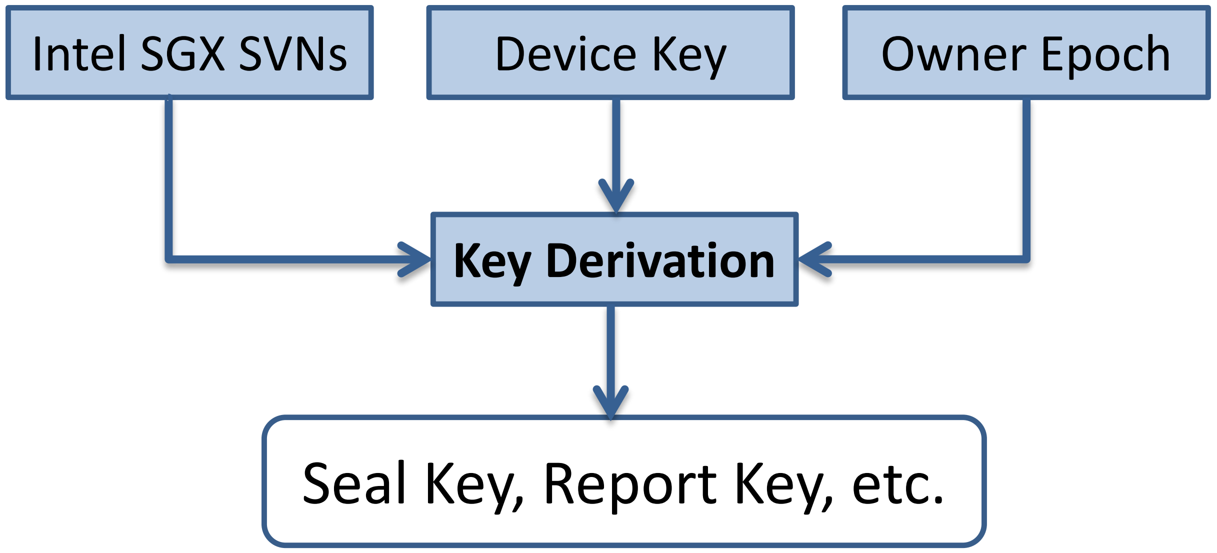
Includes:

- sealing authority **Public Key** **hash**.
- Product ID.
- SVN – Security Version Number.

EGETKEY Instruction

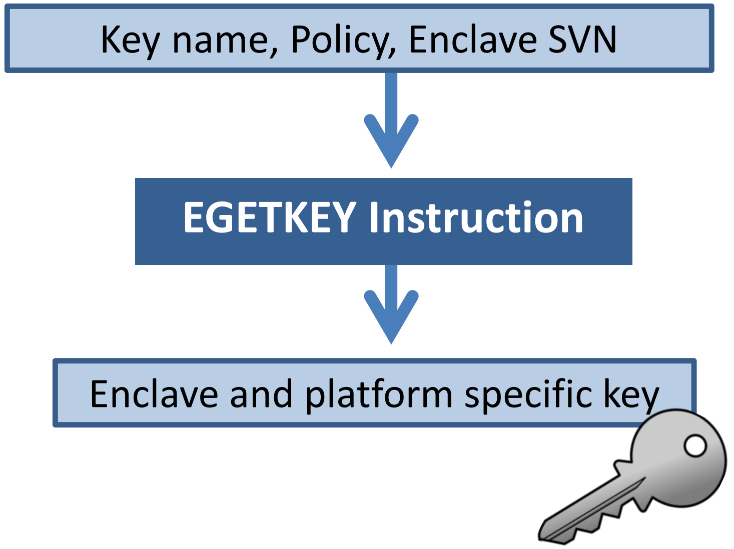


- Called by an enclave to retrieve desired key.
- Key derivation :



- Derived key is enclave and platform specific.

Sealing Process

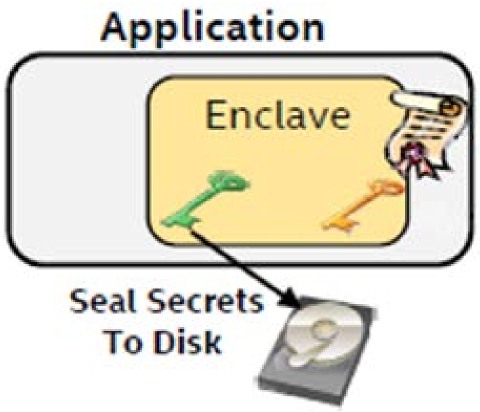

Key name, Policy, Enclave SVN
EGETKEY Instruction
Enclave and platform specific key

1. Enclave calls EGETKEY:

   Policy options are Enclave or Sealing Identity.

2. Sets sealing key SVN with key recovery transformation.

3. Seals data with yielded key and ISV chosen encryption scheme.



4. Writes sealed data to untrusted storage.

Data sealed under:

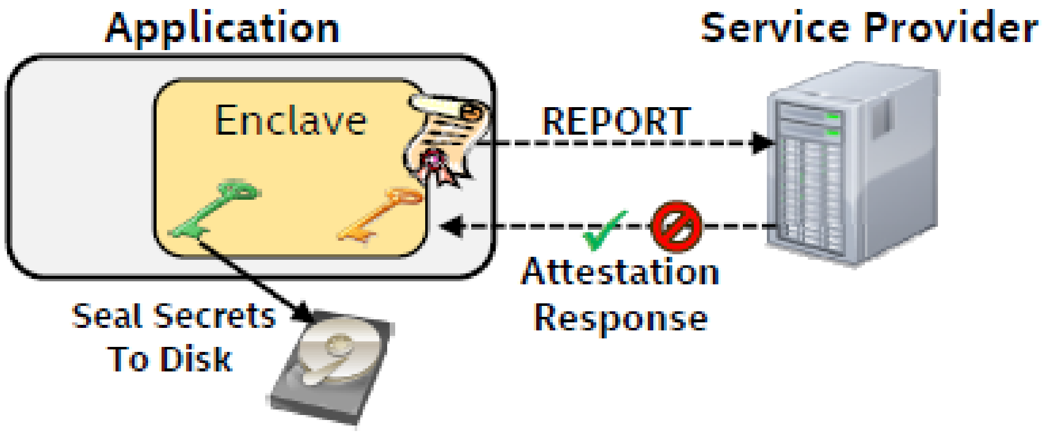
1. Processor key
2. CPU FW SVN
3. OwnerEpoch
4. Policy choice

Attestation

SGX support local and remote HW based attestation capabilities.

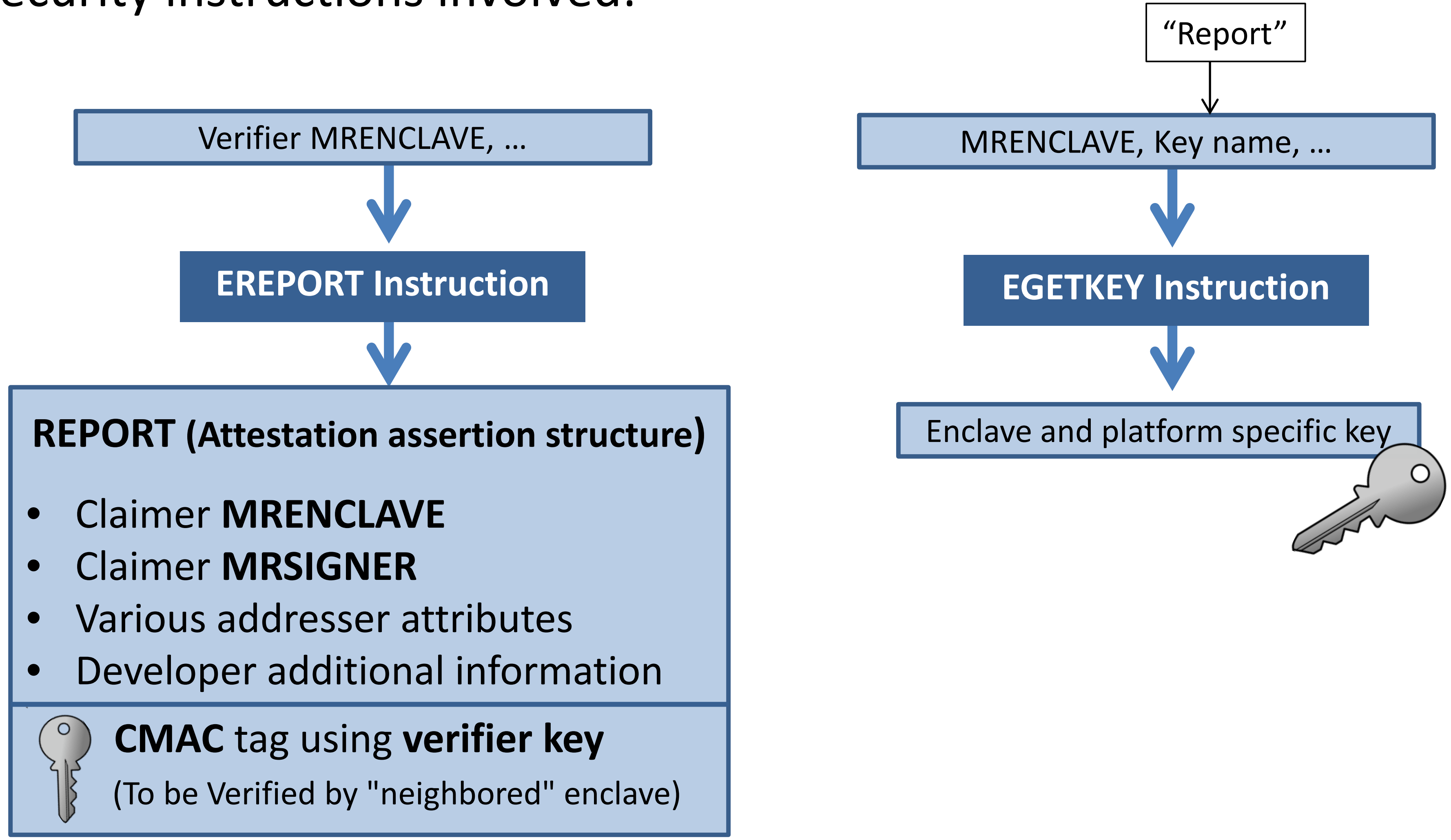
- **Local Attestation**
  Allows one enclave to attest its TCB to another enclave on the same platform.

- **Remote Attestation**
  Allows an enclave to attest its TCB to another 3-rd party entity outside of the platform.



Local Attestation

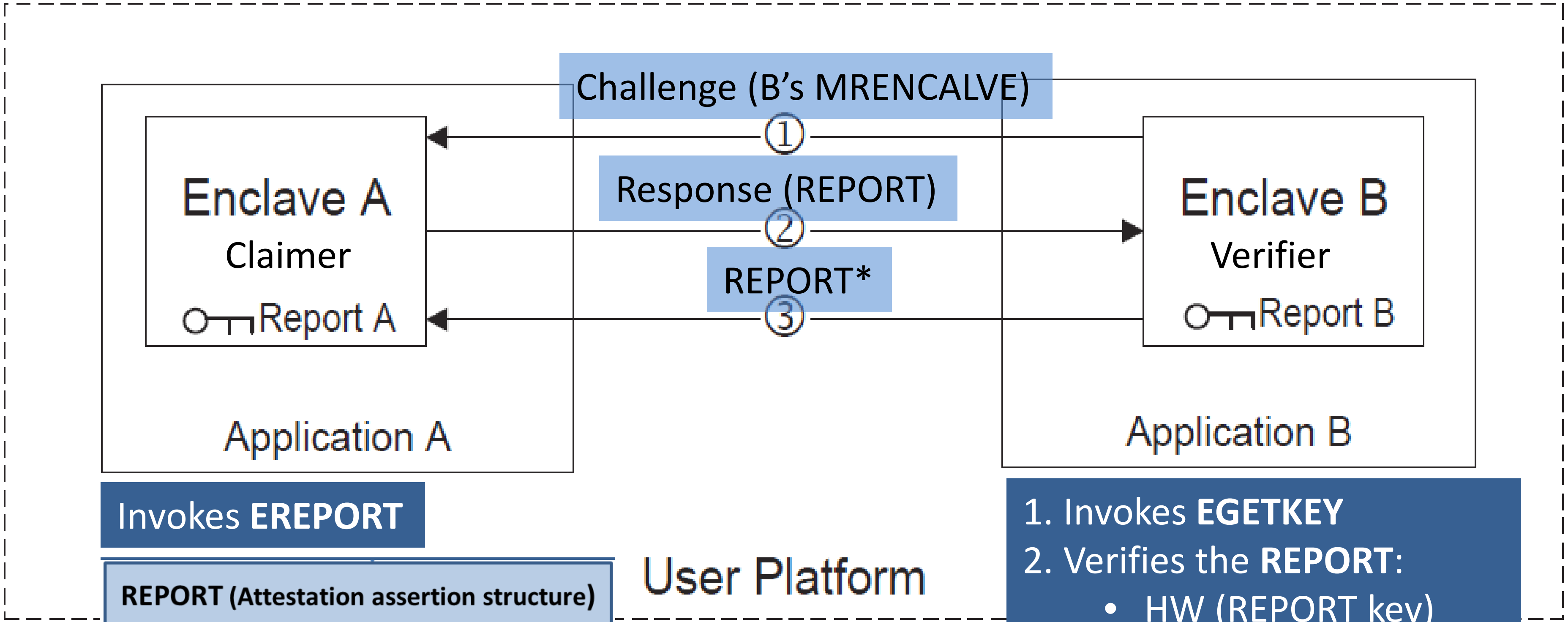
Security instructions involved:



Verifier MRENCLAVE, …
EREPORT Instruction
REPORT (Attestation assertion structure)
• Claimer MRENCLAVE
• Claimer MRSIGNER
• Various addresser attributes
• Developer additional information
CMAC tag using verifier key
(To be Verified by "neighbored" enclave)
"Report"
MRENCLAVE, Key name, …
EGETKEY Instruction
Enclave and platform specific key

Local Attestation

Protocol flow:



Invokes **EREPORT**

REPORT (Attestation assertion structure)

- Claimer **MRENCLAVE**
- Claimer **MRSIGNER**
- Various addresser attributes
- Developer additional information

CMAC tag using **verifier key**

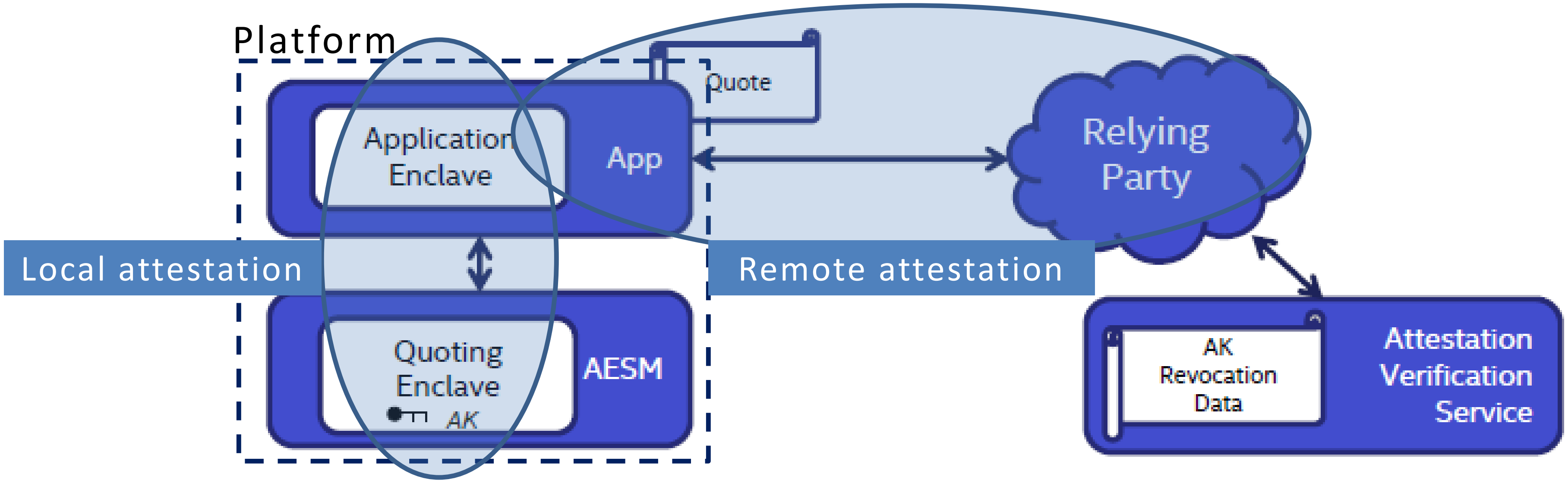
(To be Verified by "neighbored" enclave)

1. Invokes **EGETKEY**
2. Verifies the **REPORT**:
    - HW (REPORT key)
    - SW (response details)
3. Invokes **EREPORT**

* Optional

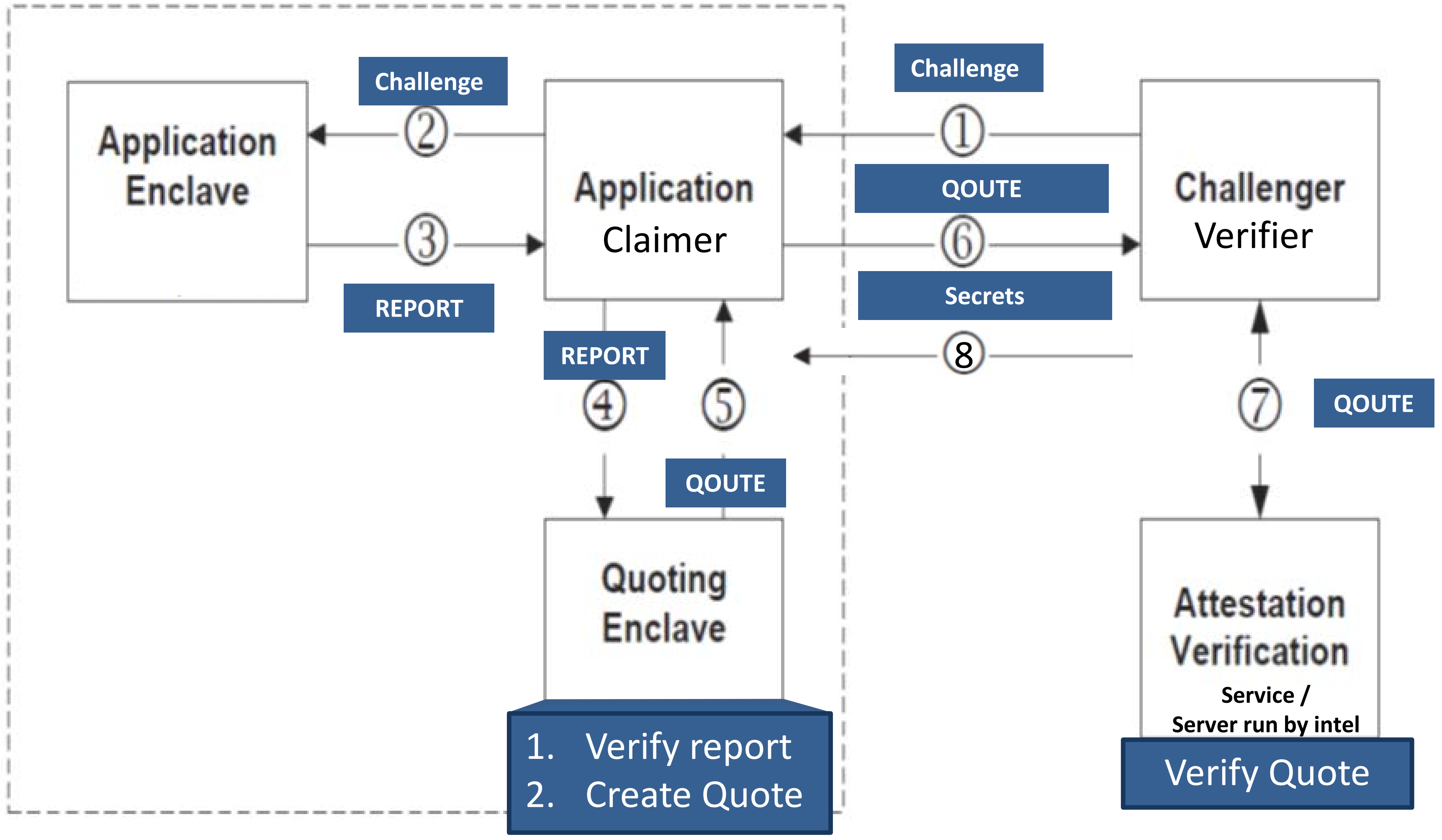
Remote Attestation

- SGX uses a **Quoting Enclave** (QE) to convert LOCAL attestations to REMOTELY verifiable assertion.
- QE locally verifies REPORT produced by Application Enclave and **signs it as a QUOTE.**
- QE uses an **asymmetric attestation key** that reflects the platforms trustworthiness.
- App sends Quote to the **Relying Party** to verify.



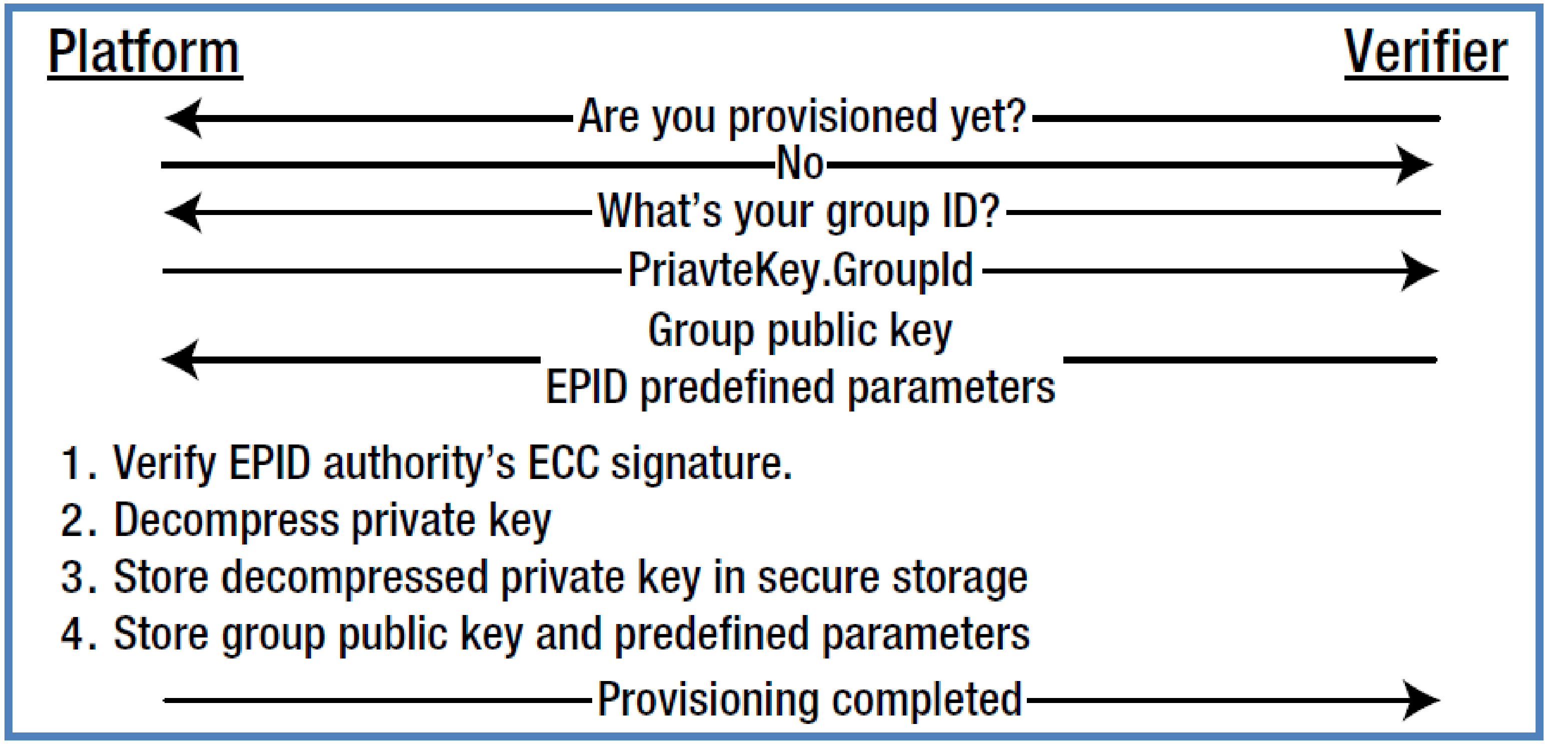
Remote Attestation

Protocol flow:



Provisioning the Attestation Key

- To save expensive secure fuse space, a unique private key is generated on manufacturing and an **unusable shorten form of it is fused** on HW.
- Group public key and other predefined parameters are sent to the device for it to extract and safely store the **functional private key** in a secure NVM.
- Integrity is checked using hardcoded **Intel EPID Authority public key**.


Platform
Verifier
Are you provisioned yet?
No
What's your group ID?
PriavteKey.GroupId
Group public key
EPID predefined parameters
1. Verify EPID authority's ECC signature.
2. Decompress private key
3. Store decompressed private key in secure storage
4. Store group public key and predefined parameters
Provisioning completed

Summary

- Enclave runs in ring-3 privilege level **deriving trust directly from hardware.**
- Developers may focus on securing a **smaller TCB**.
- Application can support **multiple enclaves**.
- Resources are run & **managed by OS**, yet protected in face of a **compromised OS/VMM**.
- **Provides integrity and confidentiality**, also in face of HW DMA attacks.

Questions & Open Discussion

- Questions ?
- What are the limitations of SGX?
- Pinpoint the root of trust assumptions.
- Techniques that can be implemented to try and breach enclave security.
- Are they any drawbacks when utilizing SGX?
- Innovative usages of SGX capabilities.