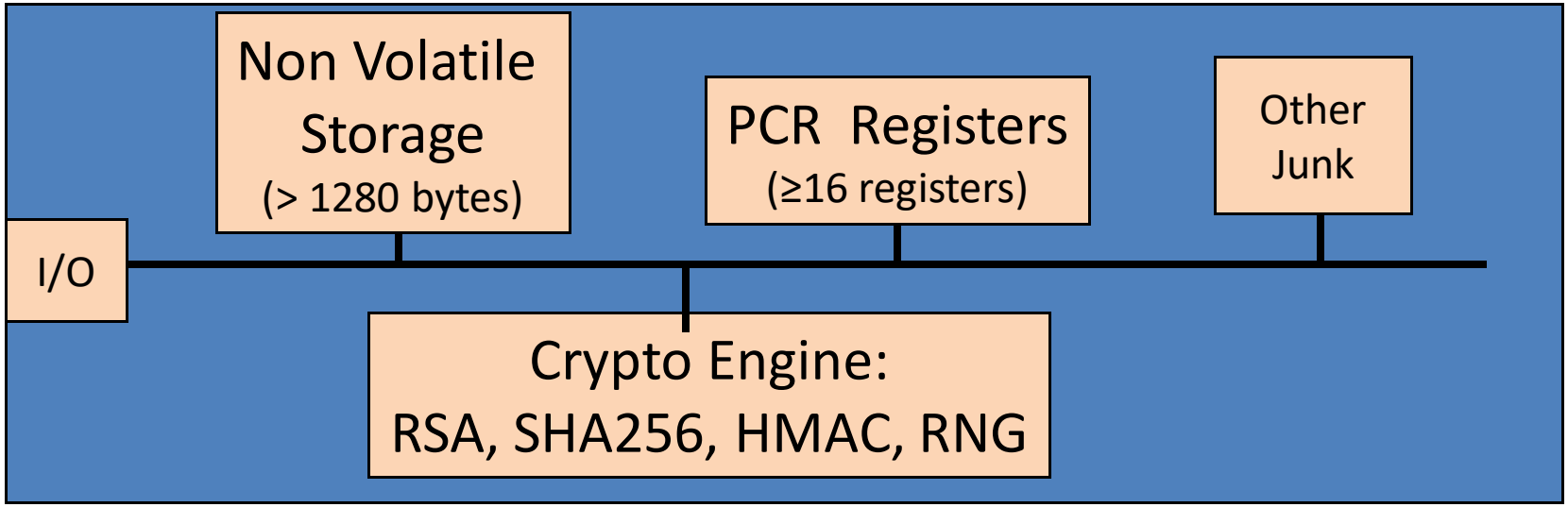
Trustworthy Computing

CS642:
Computer Security

Components on TPM chip



RSA: 1024, 2048 bit modulus

SHA256: Outputs 32 byte digest

PCR: the heart of the matter

PCR: Platform Configuration Registers

- Many PCR registers on chip (at least 16)
- Contents: 32-byte SHA256 digest (+junk)

Updating PCR #n :

- TPM_Extend(n,D): **PCR[n] ← SHA256 (PCR[n] || D)**
- TPM_PcrRead(n): returns value(PCR(n))

PCRs initialized to default value (e.g. 0) at boot time

Using PCRs: the TCG boot process (SRTM)

On power-up: TPM receives a TPM_Init signal from LPC bus.

BIOS **boot block** executes:

- Calls TPM_Startup (ST_CLEAR) to initialize PCRs to 0
  [can only be called once after TPM_Init]
- 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

The main point

After boot completes, PCR registers measure the entire
software stack that booted on the machine:

- BIOS and hardware configuration
- Boot loader and its configuration
- Operating system
- Running apps

Question from last time

- Can software verify whether it was correctly booted?

Trusted Computing

Using PCRs after boot

Using PCRs after boot

Application: **encrypted (a.k.a sealed) storage.**

Setup 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 ⟵ 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

Implementing Protected Storage

TPM_Seal: Encrypt data using RSA key on TPM. (some) Arguments:

- keyhandle: which TPM key to encrypt with
- KeyAuth: Password for using key `keyhandle'
- PcrValues: PCRs to embed in encrypted blob (named by PCR num.)
- data block: at most 256 bytes [e.g. an AES key]

Returns encrypted blob.

Main point: blob can only be decrypted with TPM_Unseal when **PCR-reg-vals = PCR-vals** in blob. TPM_Unseal fails otherwise

Protected Storage

Embedding PCR values in blob ensures that only specific apps can decrypt data.

- Changing MBR or OS kernel will change PCR values

  ⇒ data cannot be decrypted

Sealed storage: applications

Lock software on machine:

- Suppose OS and apps are sealed with MBRs PCR value
- Any changes to MBR will prevent sealed OS from loading
- Prevents modifying or inspecting OS (or loading other OS)

Web server: seal server's SSL private key

- Goal: only unmodified Apache can access SSL key
- Problem: updates to Apache or Apache config

Example: BitLocker drive encryption

tpm.msc: utility to manage TPM (e.g TakeOwnership)

- Auto generates 160-bit OwnerPassword
- Stored on TPM and in file computer_name.tpm

Volume Master Key (VMK) encrypts disk volume key

- VMK is sealed (encrypted) under TPM SRK using
  - BIOS, extensions, and optional ROM (PCR 0 and 2)
  - Master boot record (MBR) (PCR 4)
  - NTFS Boot Sector and block (PCR 8 and 9)
  - NTFS Boot Manager (PCR 10), and
  - BitLocker Access Control (PCR 11)

BitLocker

Many options for **VMK** recovery: disk, USB, paper (enc. with pwd)

- Recovery needed after legitimate system change:
  - Moving disk to a new computer
  - Replacing system board containing TPM
  - Clearing TPM (with TPM_ForceClear)

At system boot (before OS boot)

- Optional: OS loader requests PIN or USB key from user
- TPM unseals VMK, only if PCR and PIN are correct

Suppose BIOS code is updated by a firmware update.

How would the system enable access to blobs previously sealed to current BIOS version?

Patch process must re-seal all blobs with new PCR values

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
  - Verify signature SL
  - Jump to SL
- Hardware, not BIOS boot loader root of trust

Trusted Computing

Attestation

Attestation: what it does

Goal: prove to remote party what software loaded 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
- Gamers can join network only if their game 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)

- Details not important here
- AIK Private key known only to TPM
- AIK public cert issued only if EK cert is valid

Attestation: how it works

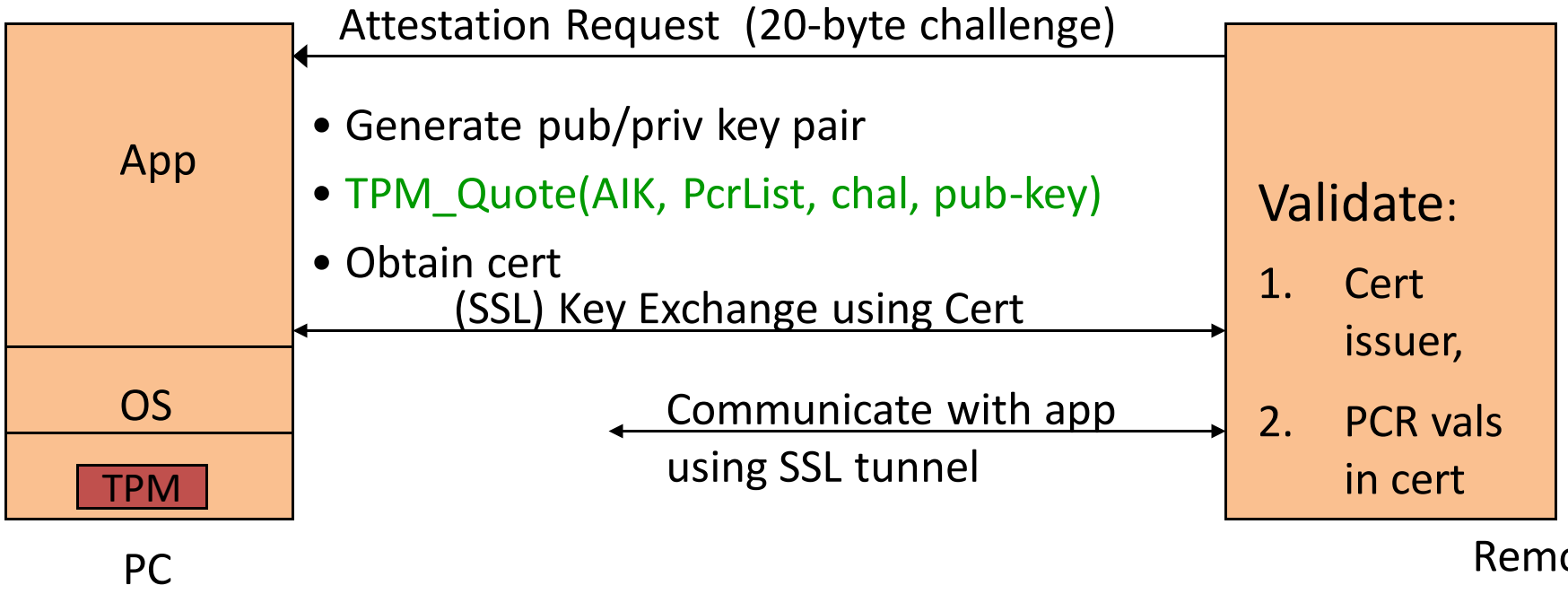
Step 2: sign PCR values (after boot) with TPM_Quote.
Arguments:

- 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.

PCR list exposes hash of what software was measured

Attestation: how it works



- Attestation typically includes key-exchange
- App must be isolated from rest of system
- **What does validator learn?**

What would go wrong if communication between app. and server were done in the clear?

User can reboot machine after attestation and run arbitrary software pretending to be app.

Trusted Computing

Attestation: challenges

1. Attesting to Current State

- Attestation only attests to what code was loaded.
- Does not say whether running code has been compromised.
  - Problem: what if Quake vulnerability exploited after attestation took place?
- Can we attest to the current state of a running system?
  - … or is there a better way?

2. Encrypted viruses

Suppose malicious music file exploits bug in video player.

- Video file is encrypted.
- TPM prevents anyone from getting video file in the clear.
- Can anti-virus companies study virus without seeing its code in the clear?
- How would you solve this?

3. TPM Compromise

Suppose one TPM Endorsement Private Key is exposed

- Destroys all attestation infrastructure:
  - Embed private EK in TPM emulator.
  - Now, can attest to anything without running it.

⇒ Certificate Revocation is critical for Attestation.

Intel SGX

SGX: Goals

Extension to Intel processors that support:

- **Enclaves**: running code and memory isolated from the rest of system
- **Attestation**: prove to local/remote system what code is running in enclave
- **Minimum TCB**: only processor is trusted
  nothing else: DRAM and peripherals are untrusted
  ⇒ all writes to memory must be encrypted
  ⇒ OS/Hypervisor not trusted

Like using TPM, but can switch back/forth to untrusted execution

Applications

Server side:

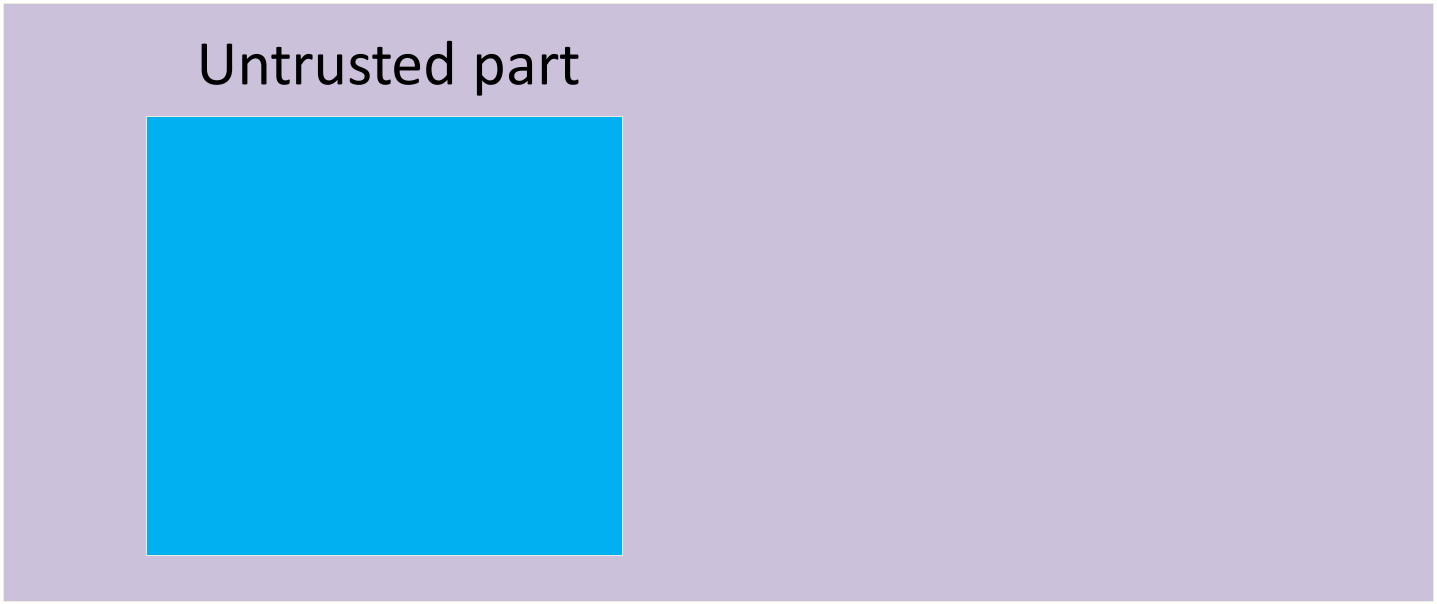
- Storing a Web server HTTPS secret key
  secret key only opened inside of an enclave
  malware cannot get the key
- Running a private job in the cloud: job runs in enclave
  Cloud admin cannot get code or data of job

Client side:

- Hide anti-virus (AV) signatures:
  AV signatures are only opened inside an enclave
  not exposed to adversary in the clear

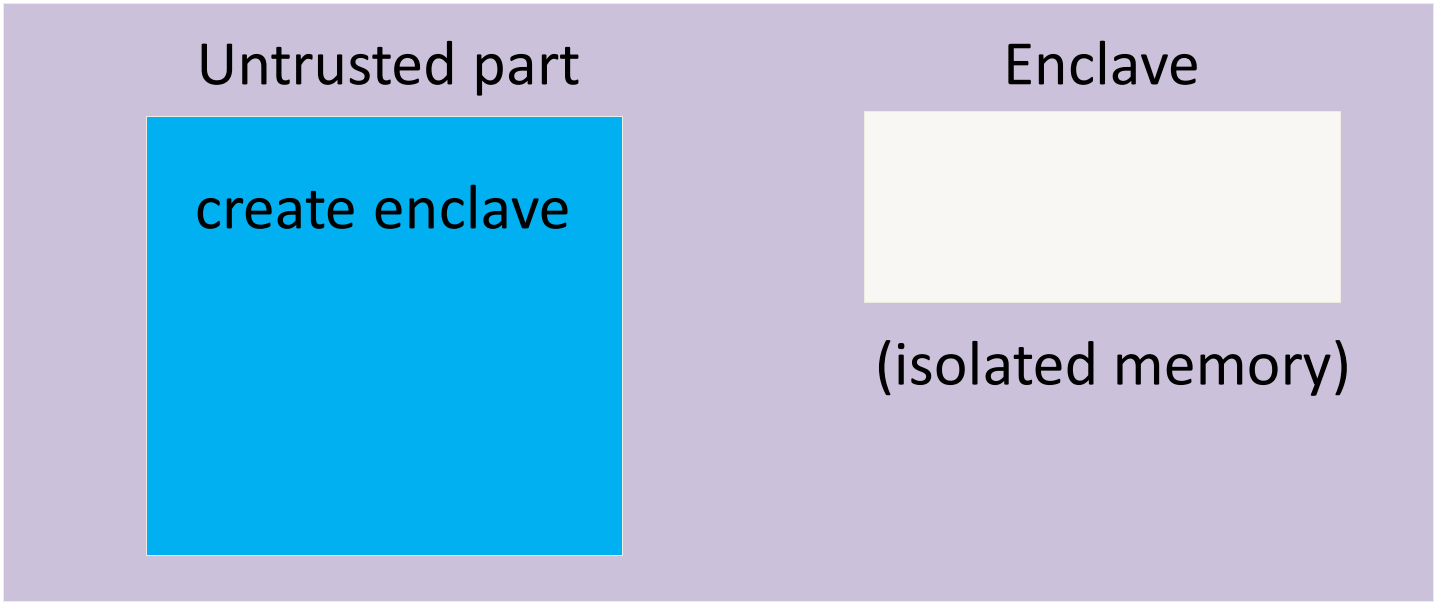
How does it work?

An application defines part of itself as an enclave



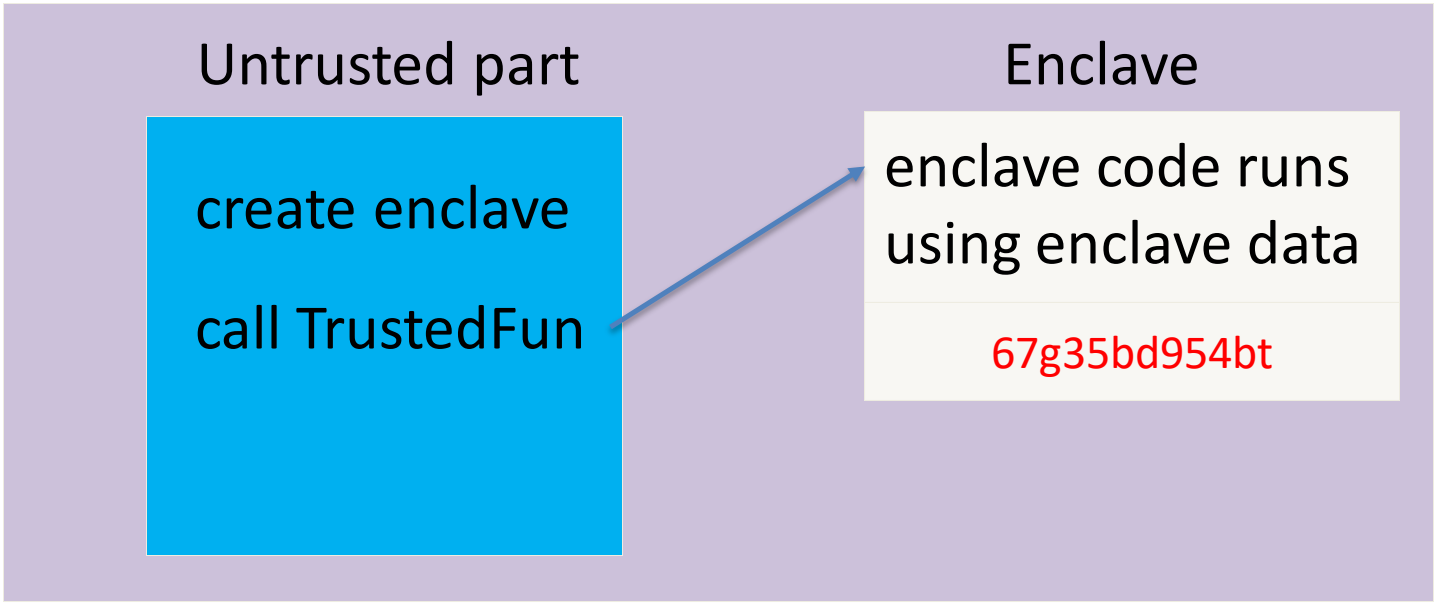
How does it work?

An application defines part of itself as an enclave



How does it work?

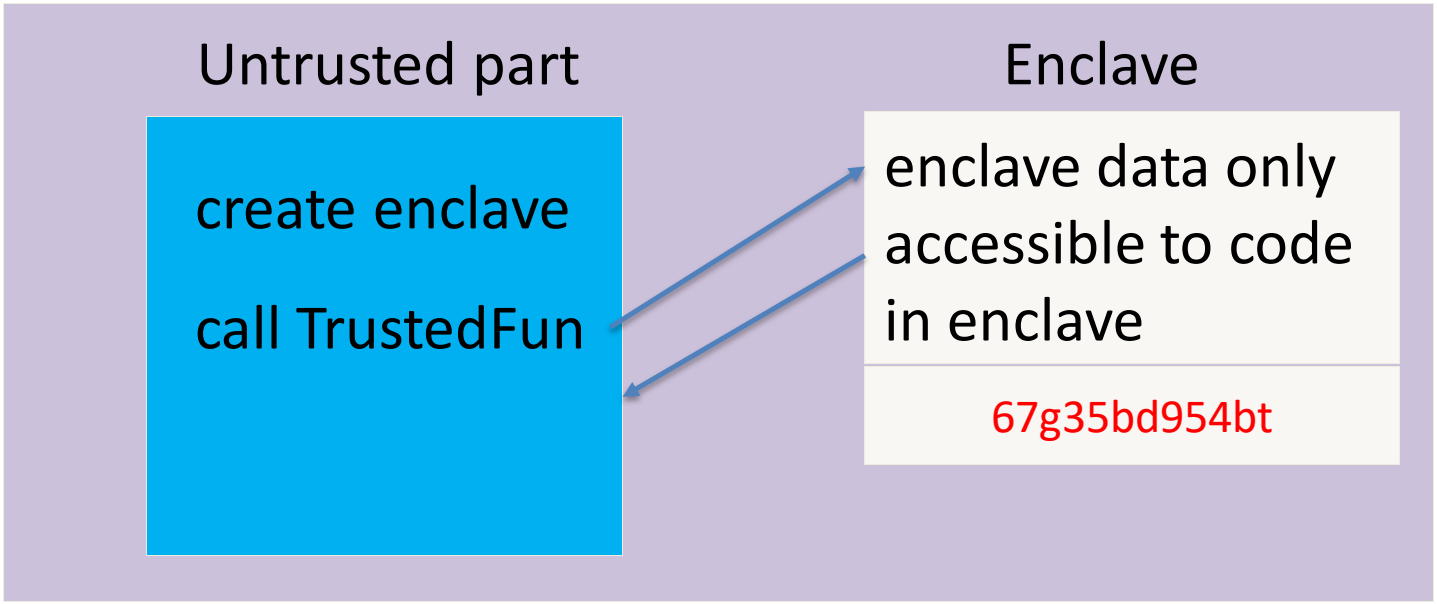
An application defines part of itself as an enclave



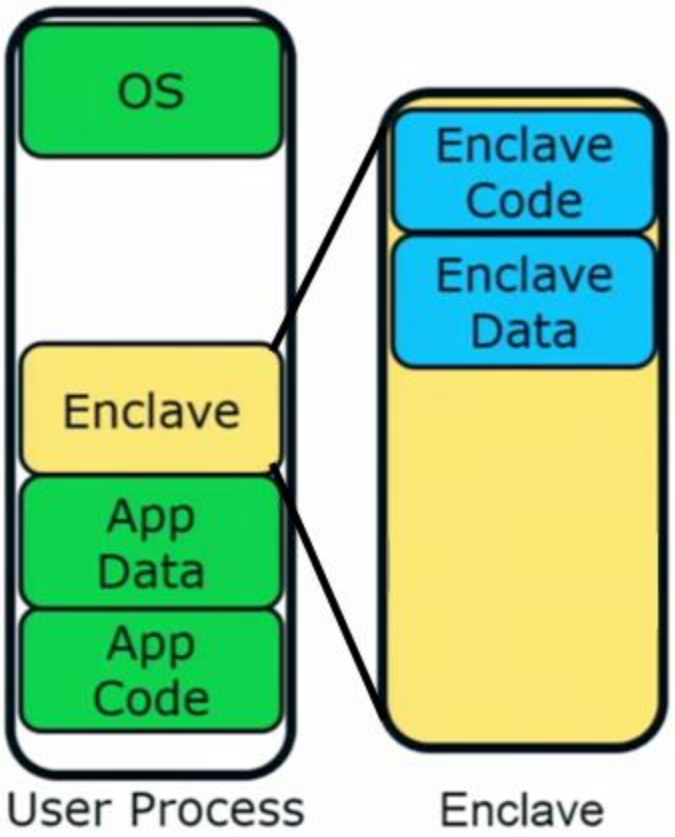
Application

How does it work?

An application defines part of itself as an enclave

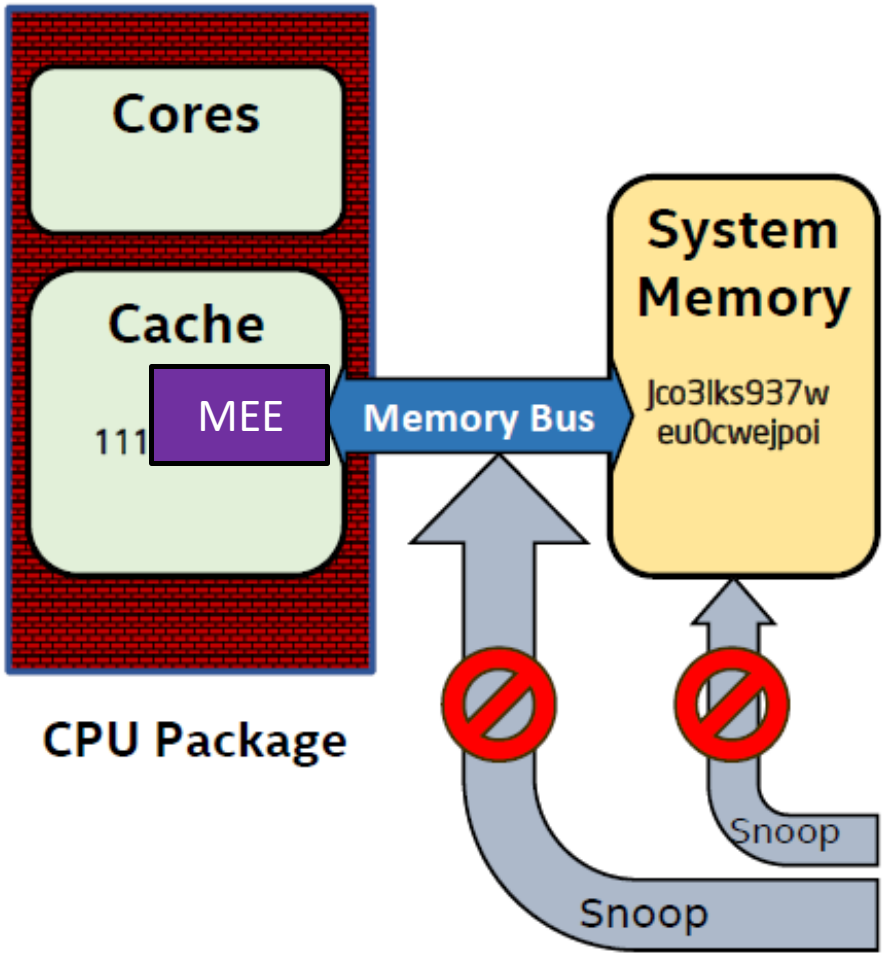


The Enclave – A trusted execution environment embedded in a process



- 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

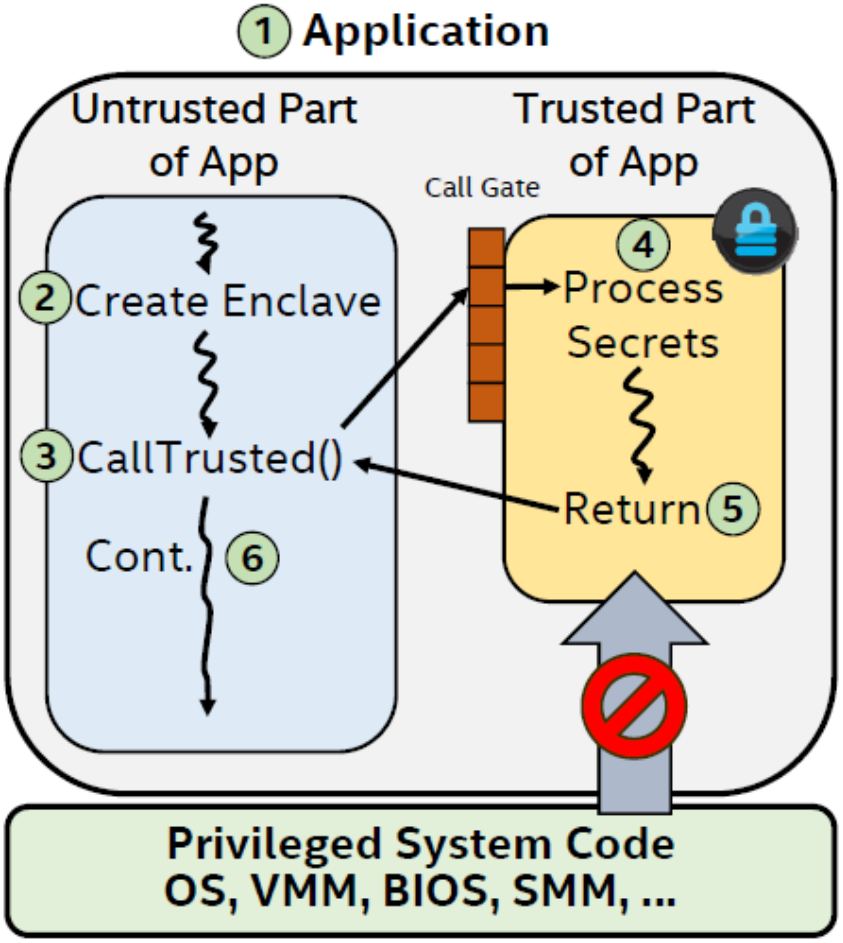


- Security perimeter is the CPU package boundary.
- Data and code unencrypted inside CPU package.
- Data and code outside CPU is encrypted and integrity checked.
  - Registers encrypted on trap
  - Memory encrypted when leaves cache

- 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.

Creating an enclave: new instructions

- **ECREATE**: establish memory address for enclave
- **EADD**: copies memory pages into enclave
- **EEXTEND**: computes hash of enclave contents (256 bytes at a time)
- **EINIT**: verifies that hashed content is properly signed
  if so, initializes enclave (signature = RSA-3072)

- **EENTER**: call a function inside enclave
- **EEXIT**: return from enclave

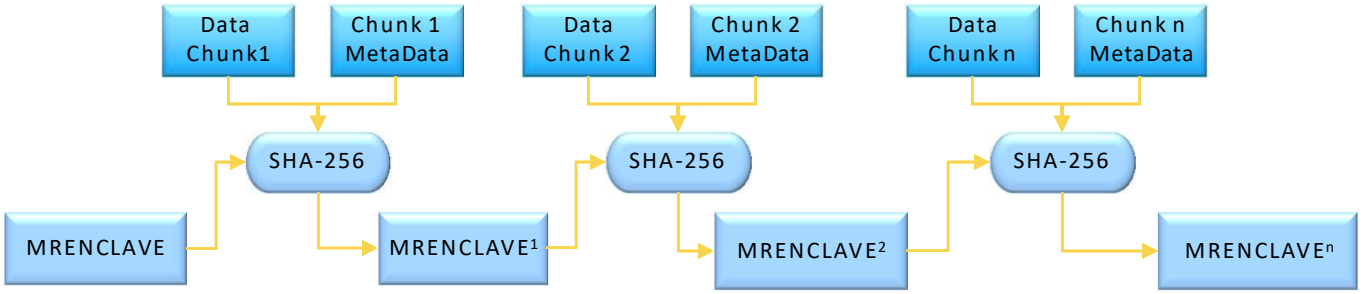
Enclave Measurement

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

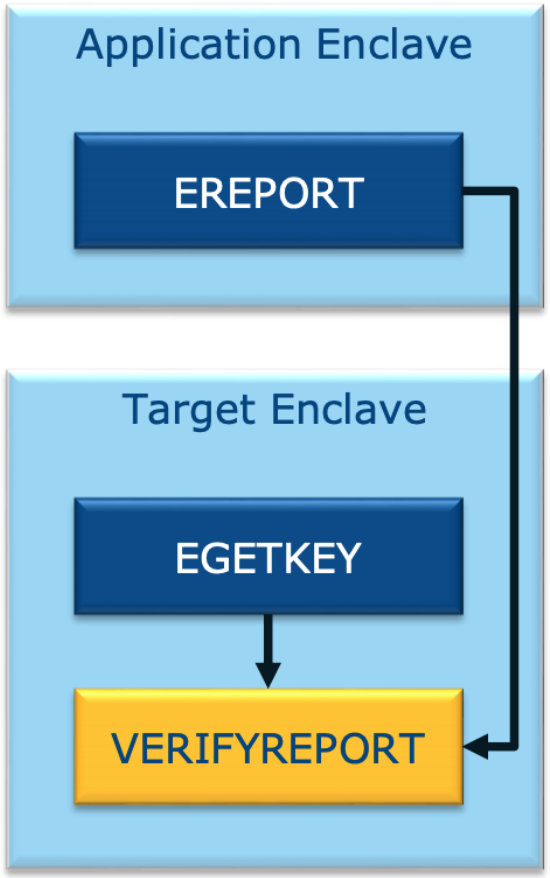
MRENCLAVE ("Enclave Identity") is a 256-bit digest of the log

- Represents the enclave's software TCB



Attestation

- Enclave uses EREPORT instruction to construct a hardware-based assertion describing the enclave's identity
  - Enclave attributes
  - Enclave measurement
  - User-supplied data (e.g., public key)
- EREPORT is parameterized by the desired target of the attestation
  - Provided by calling enclave
  - Report structure is secured by EREPORT using the report key of the target enclave
- Target enclave uses its report key to verify the report structure (in software)

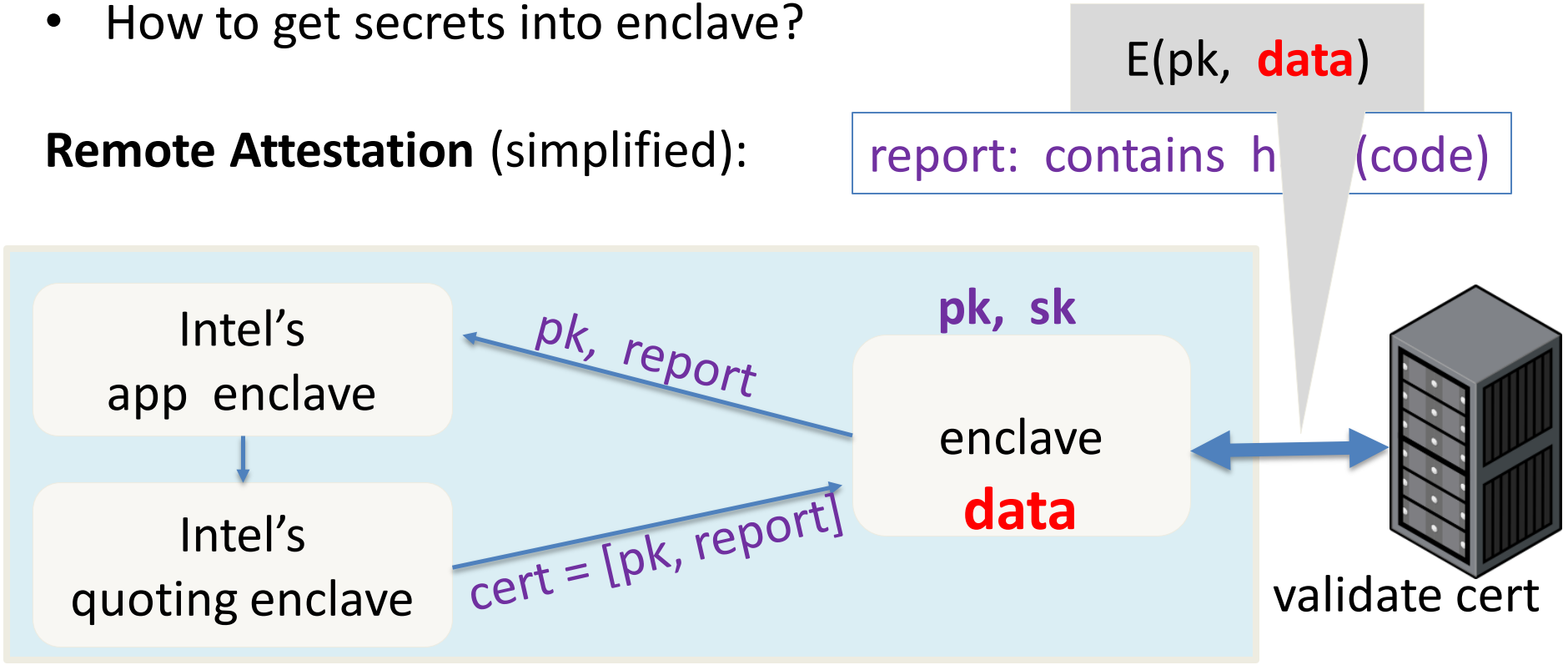


Provisioning enclave with secrets: attestation

The problem: enclave is in the clear prior to activation (EINIT)

- How to get secrets into enclave?

Remote Attestation (simplified):



Memory Encryption Engine (MEE)

- Resides within the CPU as a Memory Controller extension.
- Encrypts, decrypts page swaps and integrity checks them , providing data confidentiality and integrity.
- TRNG generates at boot time separate keys for encryption and integrity.
- Keys are held in 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… Limits enclave to 128MB

Summary

SGX: a powerful architecture for managing secret data

- Enables processing of data that cannot be read by anyone, except for code running in enclave
- Minimal TCB: nothing trusted except for x86 processor
- Hard to use with legacy applications