Hardware Security Module

Mengjia Yan

Spring 2024

Secure Processors/HSM

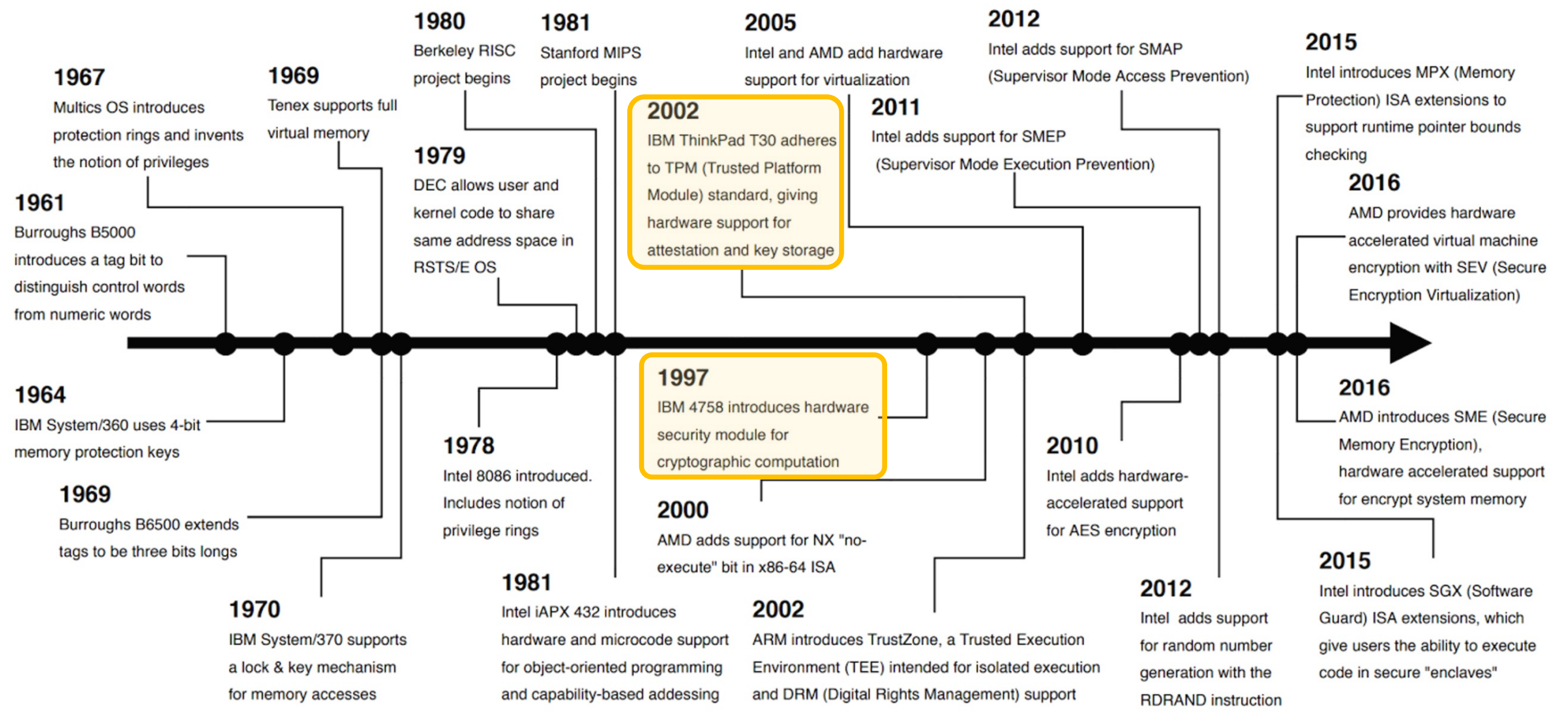
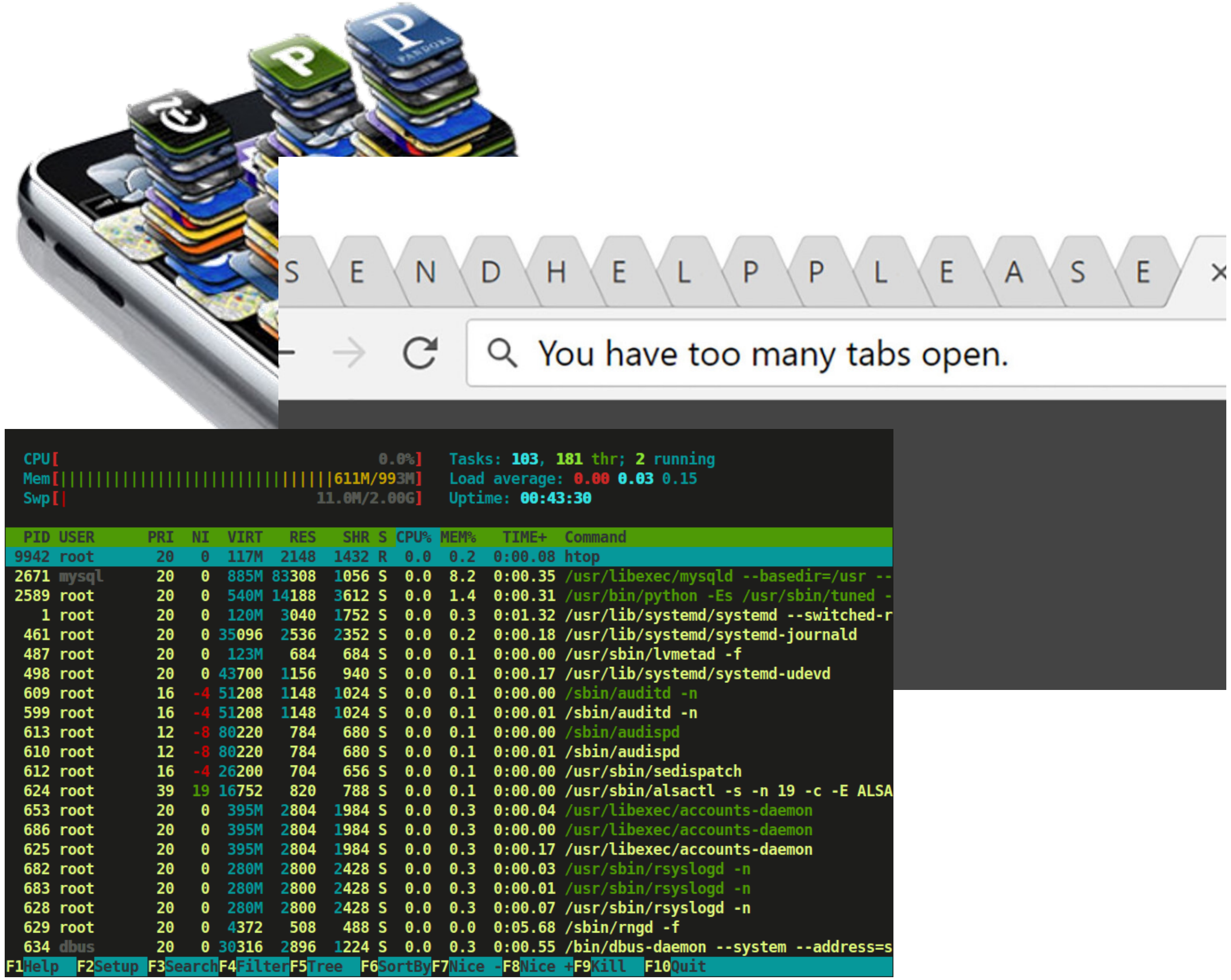


Figure 1: Sixty years of hardware support for security

Apple Secure Enclave

Introduction to Security for Computer Architecture Students. Adam Hastings, Mohammed Tarek, Simha Sethumadhavan.
https://www.cs.columbia.edu/~simha/ch1_supplement.pdf

Security Contexts #1

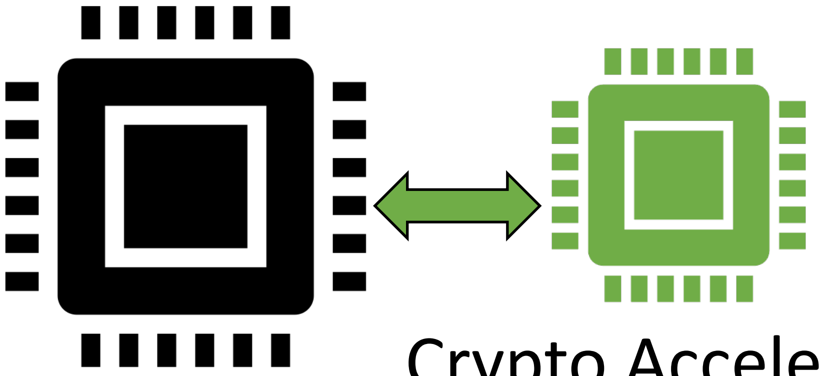


- Software can be buggy (or sometimes malicious)

- Running daily applications together with security-sensitive applications

- Can we do better than software-based isolation?

Before IBM 4758 (1999): Crypto Accelerators

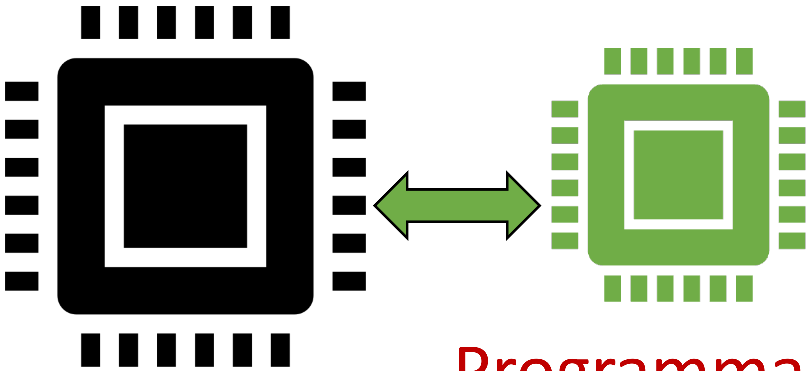


CPU

Crypto Accelerators
(AES, RSA, etc…)

IBM 4758 (1999) -- 4765 (2012)

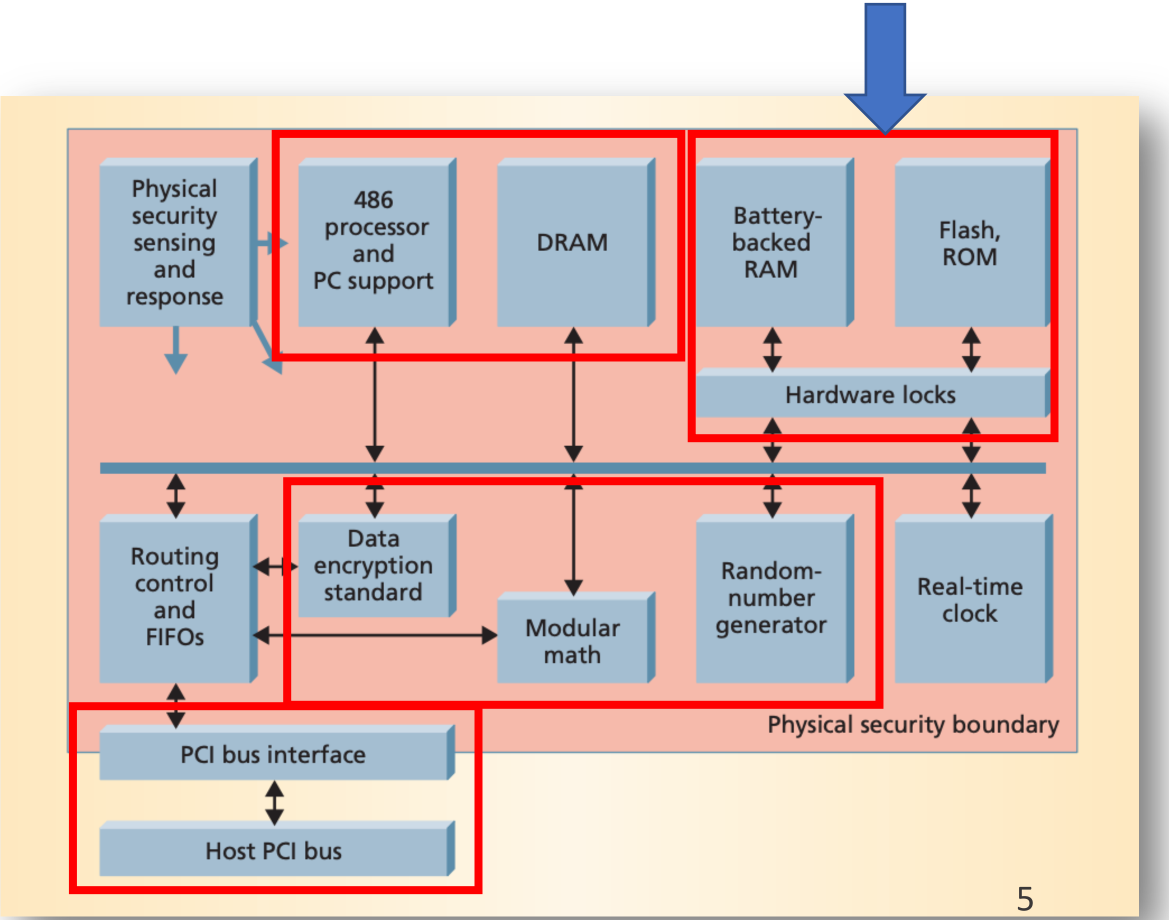
- Goal: a programmable, secure co-processor.
- High level idea: virtual locker room



CPU

Programmable
Secure
Co-processor

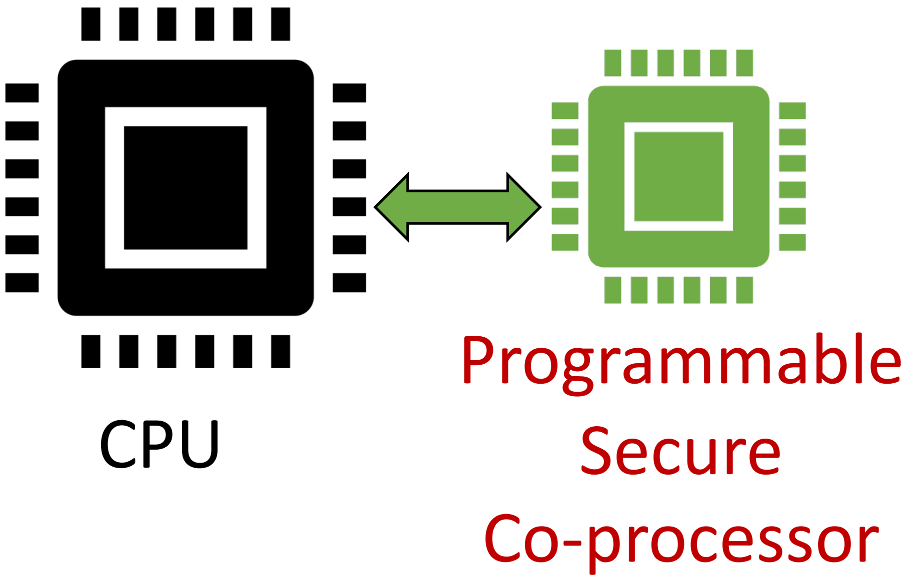
Stores the firmware
and secret keys



Why this is more secure?

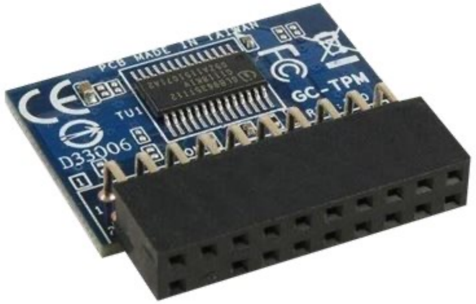
This assumes software-based attacks. What about physical attacks?

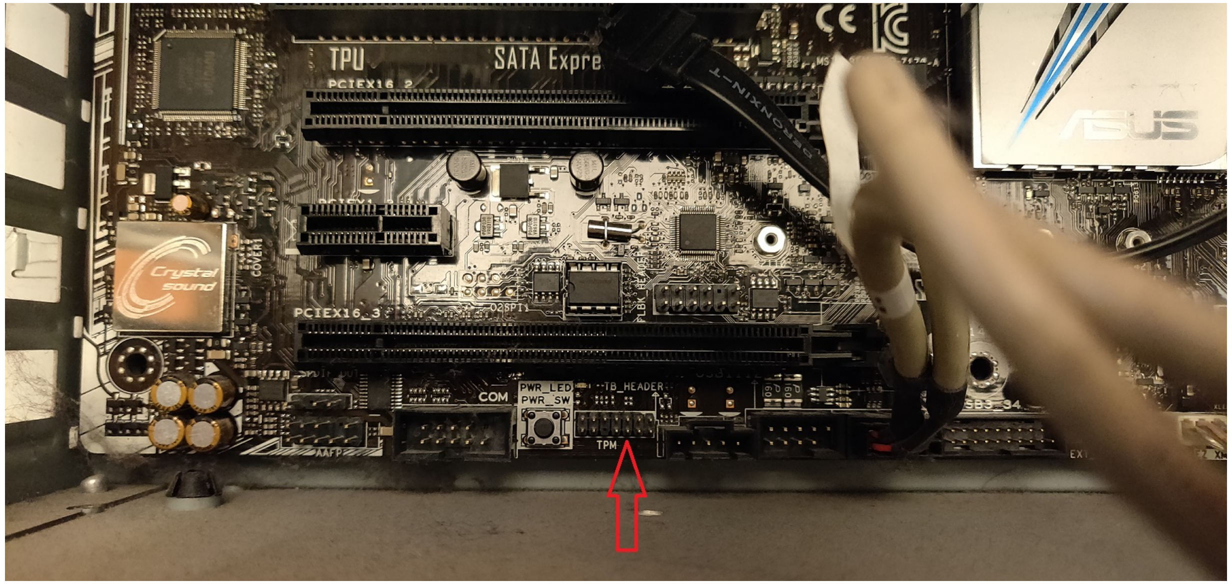
- Physical isolation (Not share physical memory)
- Narrow interface, only interact with external worlds via APIs (keys do not leave the co-processor)
- Simpler software on co-processor, so fewer bugs (maybe can be formally verified)
- Problem?
  - The SWOFTWARE! Bad programmability.



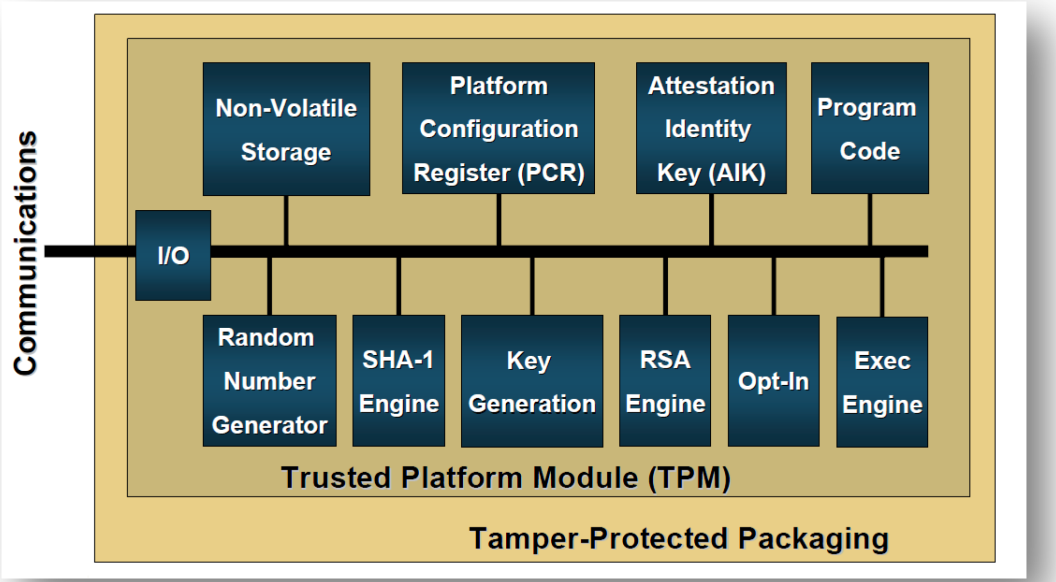
Trusted Platform Module (TPM)

- "*Commoditized* IBM 4758": Standard LPC interface attaches to commodity motherboards



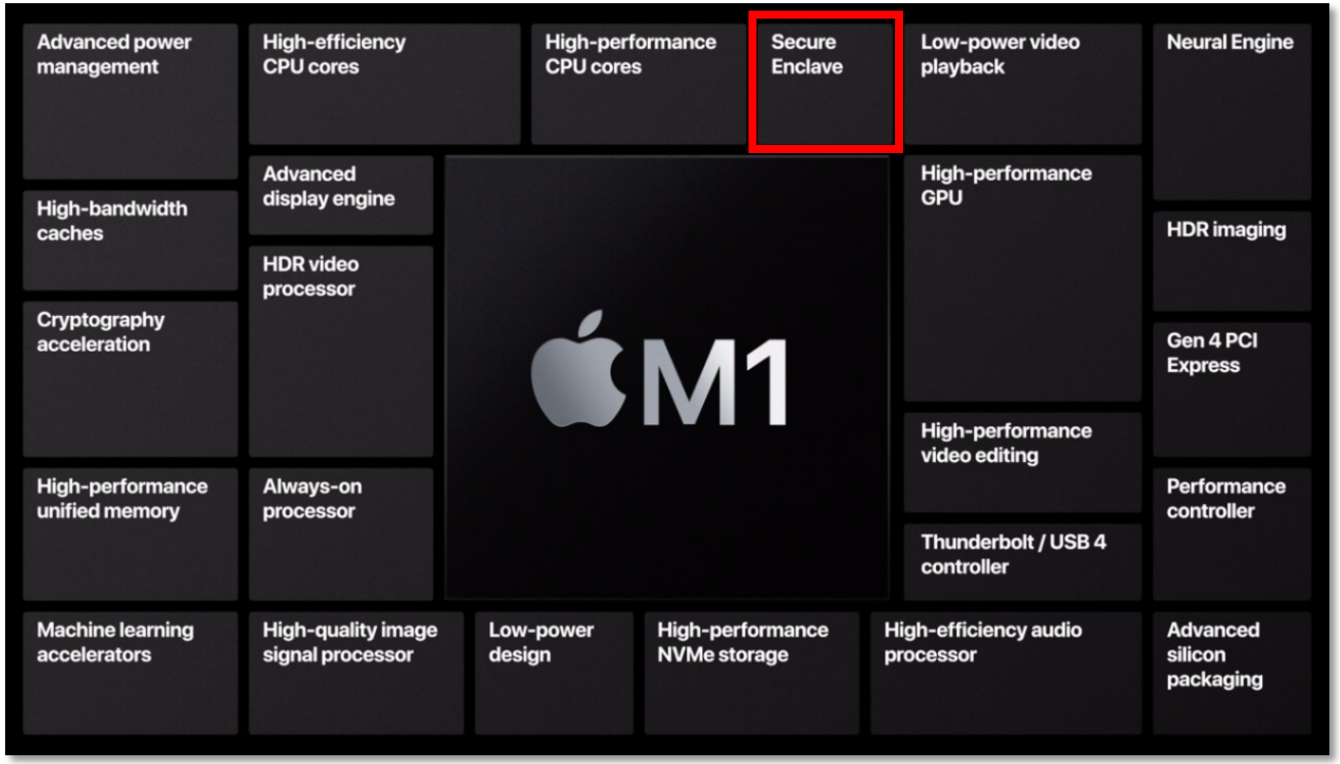


https://scotthelme.co.uk/upgrading-my-pc-with-a-tpm/



Apple Secure Enclave

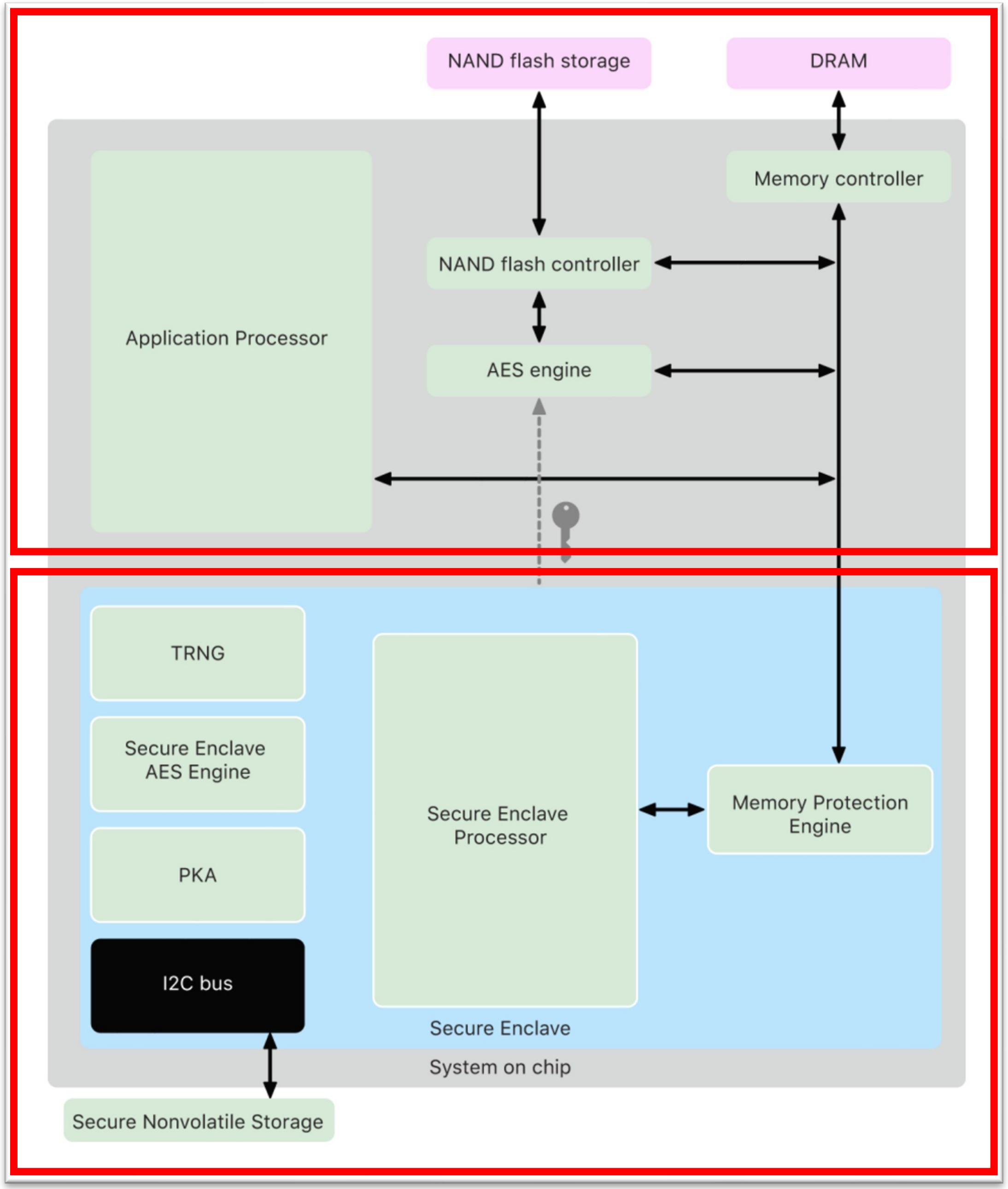
- Advantage: one company controls both the hardware and the software
- Apple secure enclave runs a customized formally verified micro-kernel OS



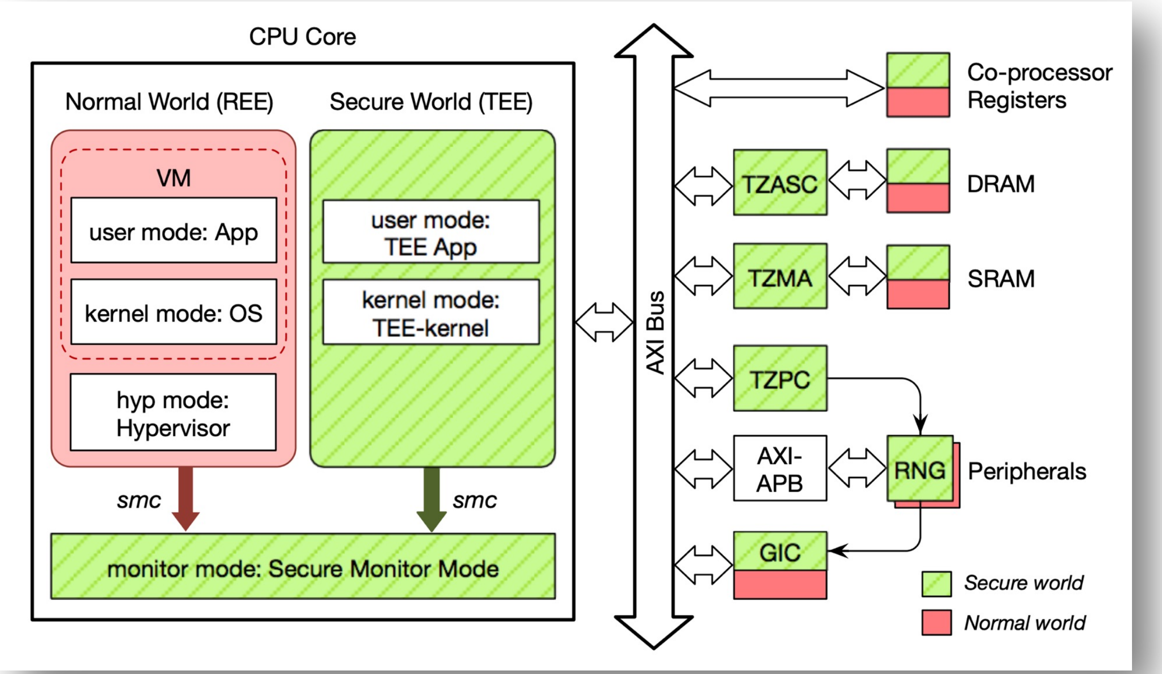
Separate Cores

The secure enclave:

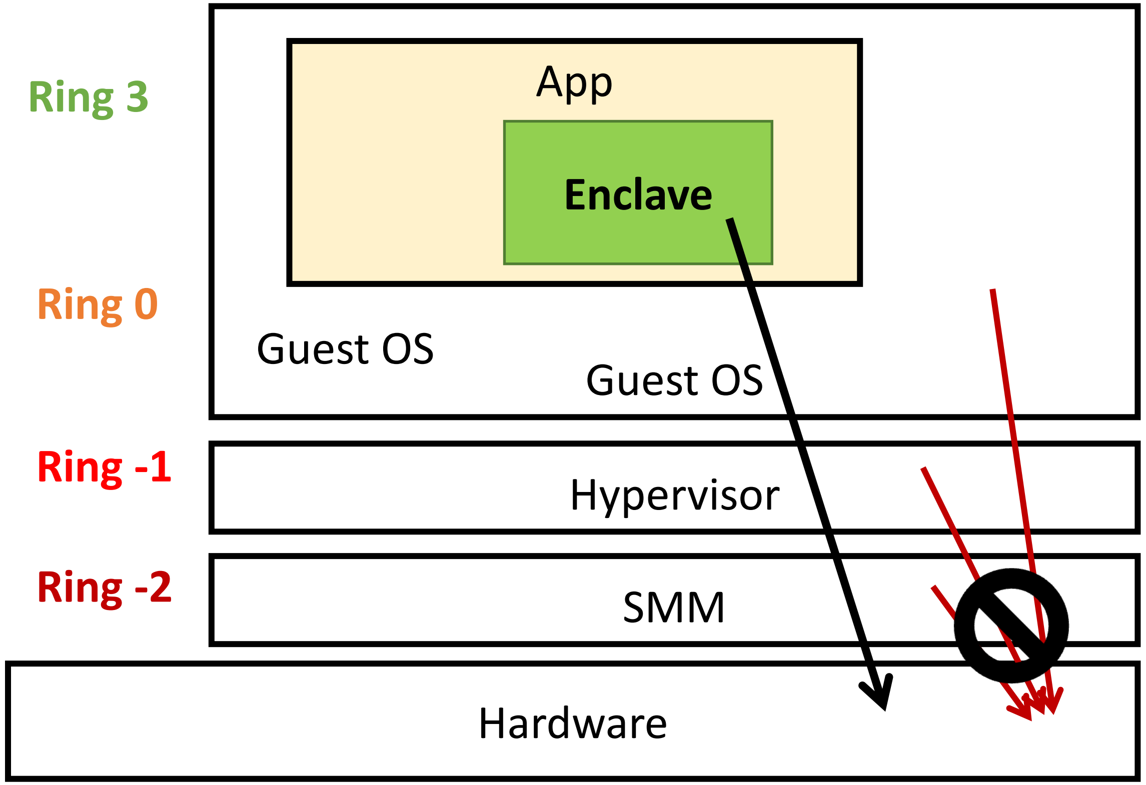
- Not general-purpose, only run secure enclave functionality, no user code
- Block vulnerabilities due to software bugs (running L4 microkernel) and side channels



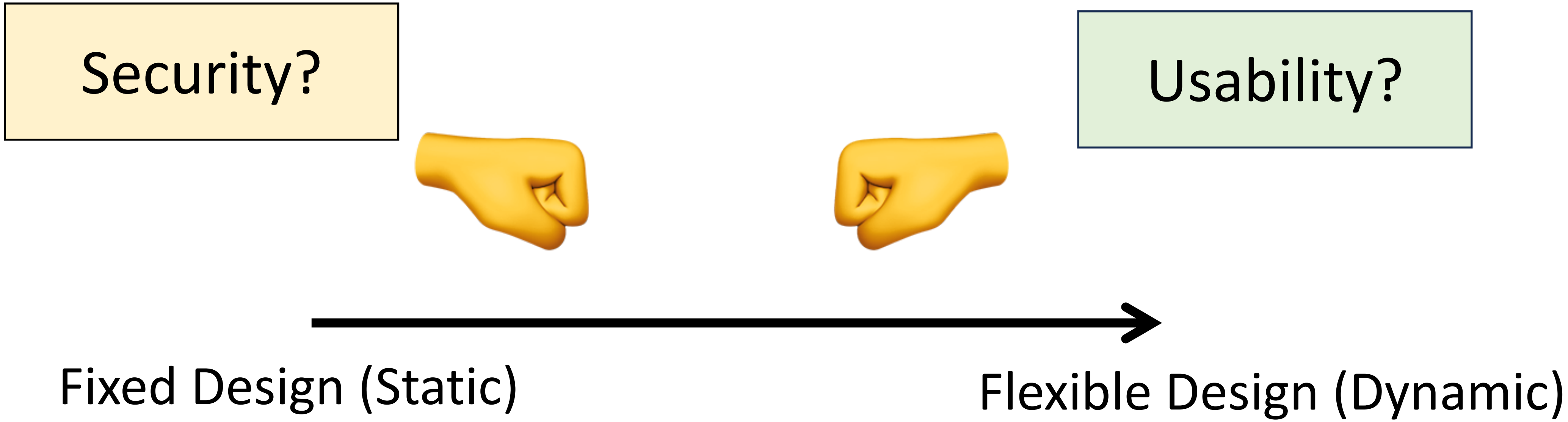
The Trends (isolation with some sharing?)



ARM TrustZone



Intel SGX model


Security?
Usability?
Fixed Design (Static)
Flexible Design (Dynamic)

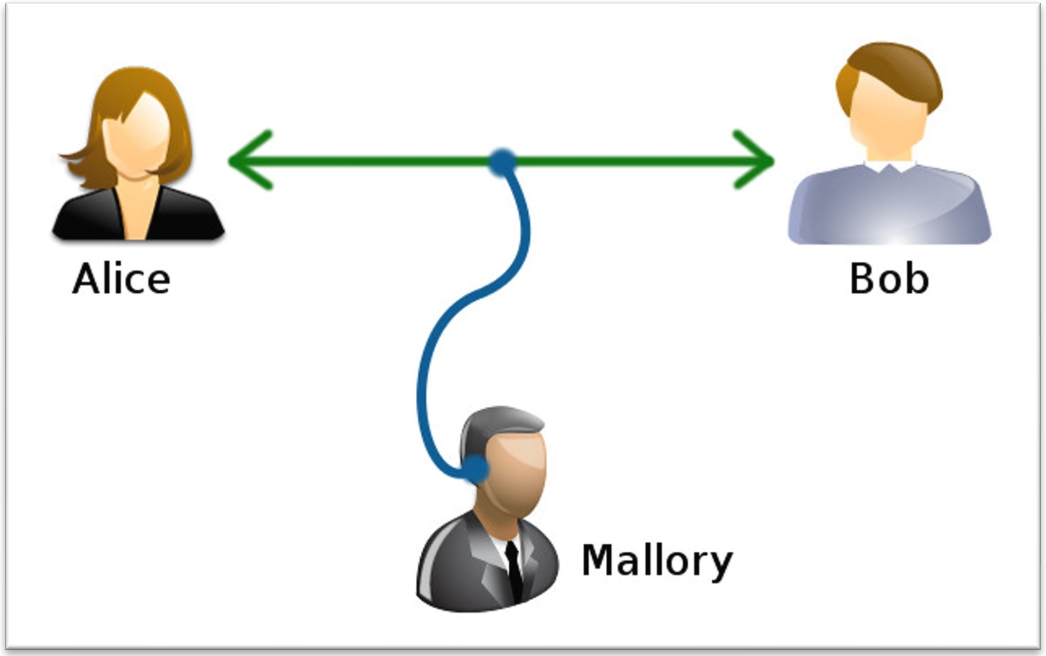
Security Contexts #2

- Disk lost or removed, leading to confidentiality leakage.
- Data encryption with weak passwords, such as, 6-digit passcode.

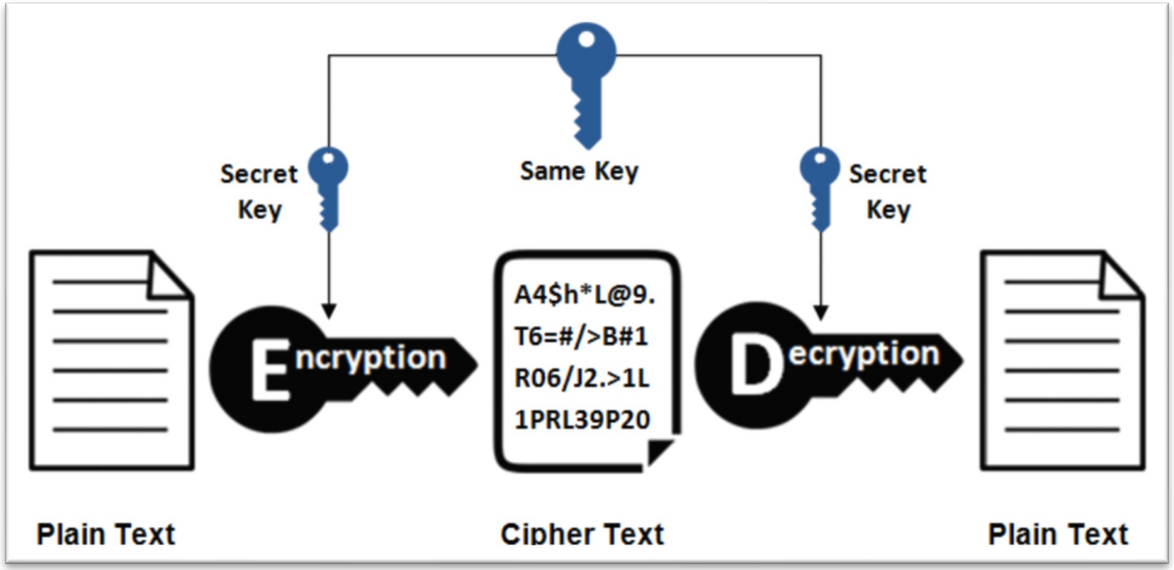
Bind data/application with hardware using crypto.

Security Property and Crypto Primitives

- Confidentiality
  - Symmetric
  - Asymmetric
- Integrity
- Freshness



Symmetric Cryptography



- One-time-pad (OTP)

Encryption:
`ciphertext = key ⊕ plaintext`

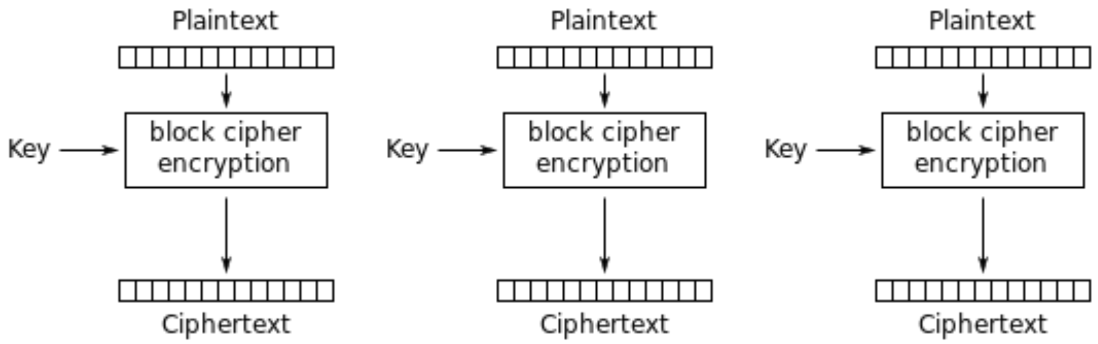
Decryption:
`plaintext = key ⊕ ciphertext`

How about encrypting arbitrary length message? Any problems?

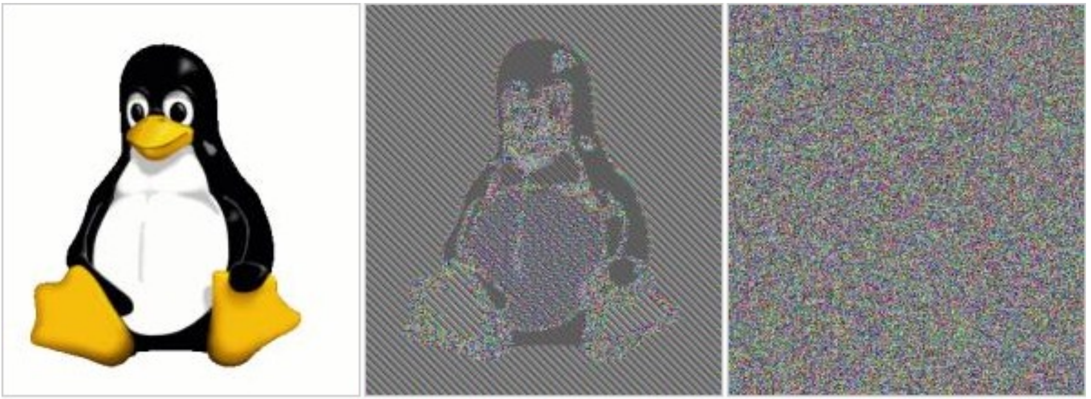
Block ciphers (e.g., DES, AES)

- Divide data in blocks and encrypt/decrypt each block
- AES block size can be 128, 192, 256 bits

ECB IS NOT RECOMMENDED

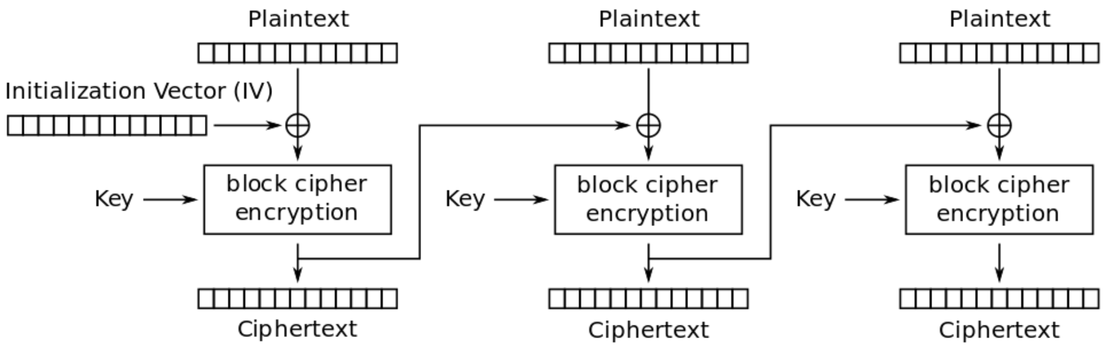


Electronic Codebook (ECB) mode encryption

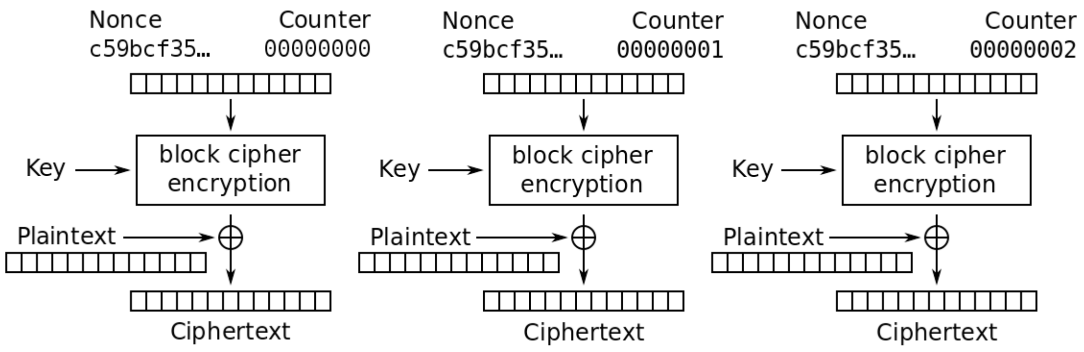


Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

Other Block cipher Modes



Cipher Block Chaining (CBC) mode encryption



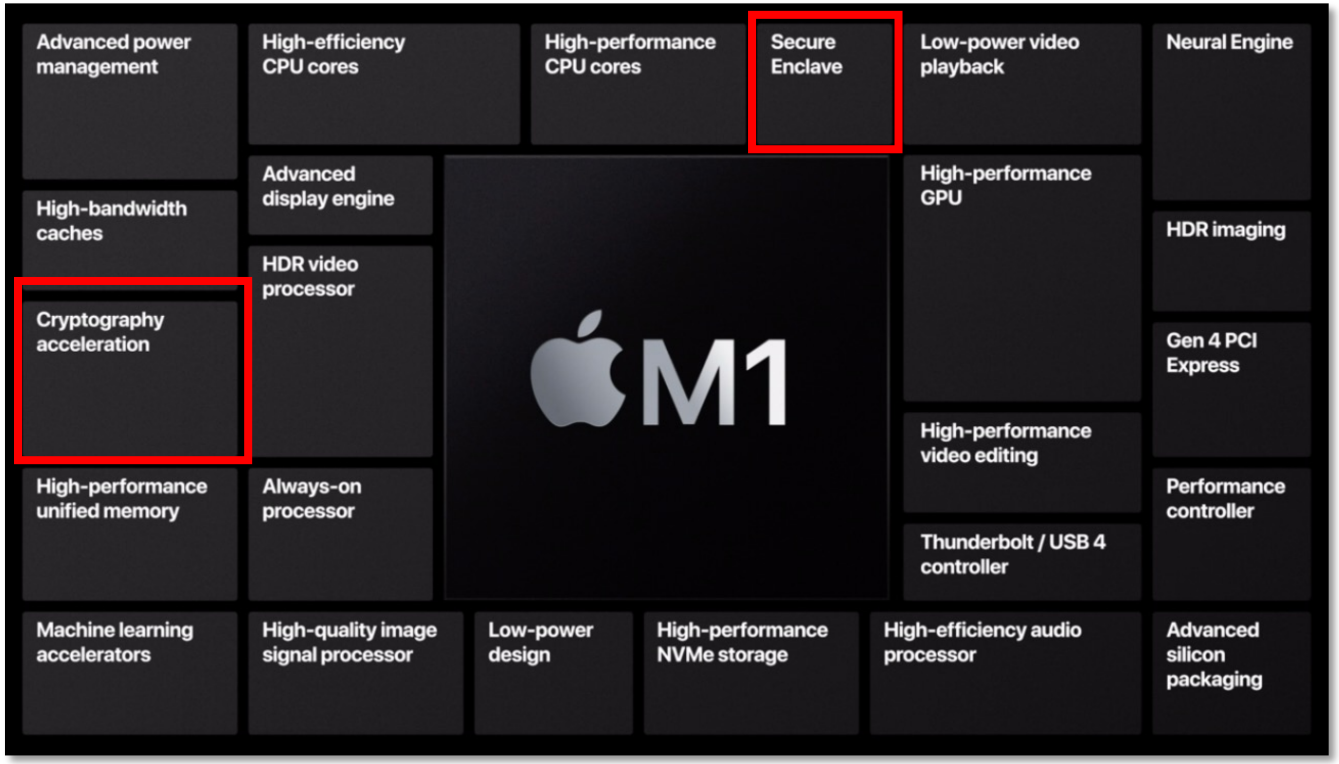
Counter (CTR) mode encryption

`IV` can be public, but need to ensure to not reuse `IV` for the same key.

- Real-world application: file/disk encryption and memory encryption.
- How to exchange the shared key between two parties?

Diffie-Hellman
Key Exchange

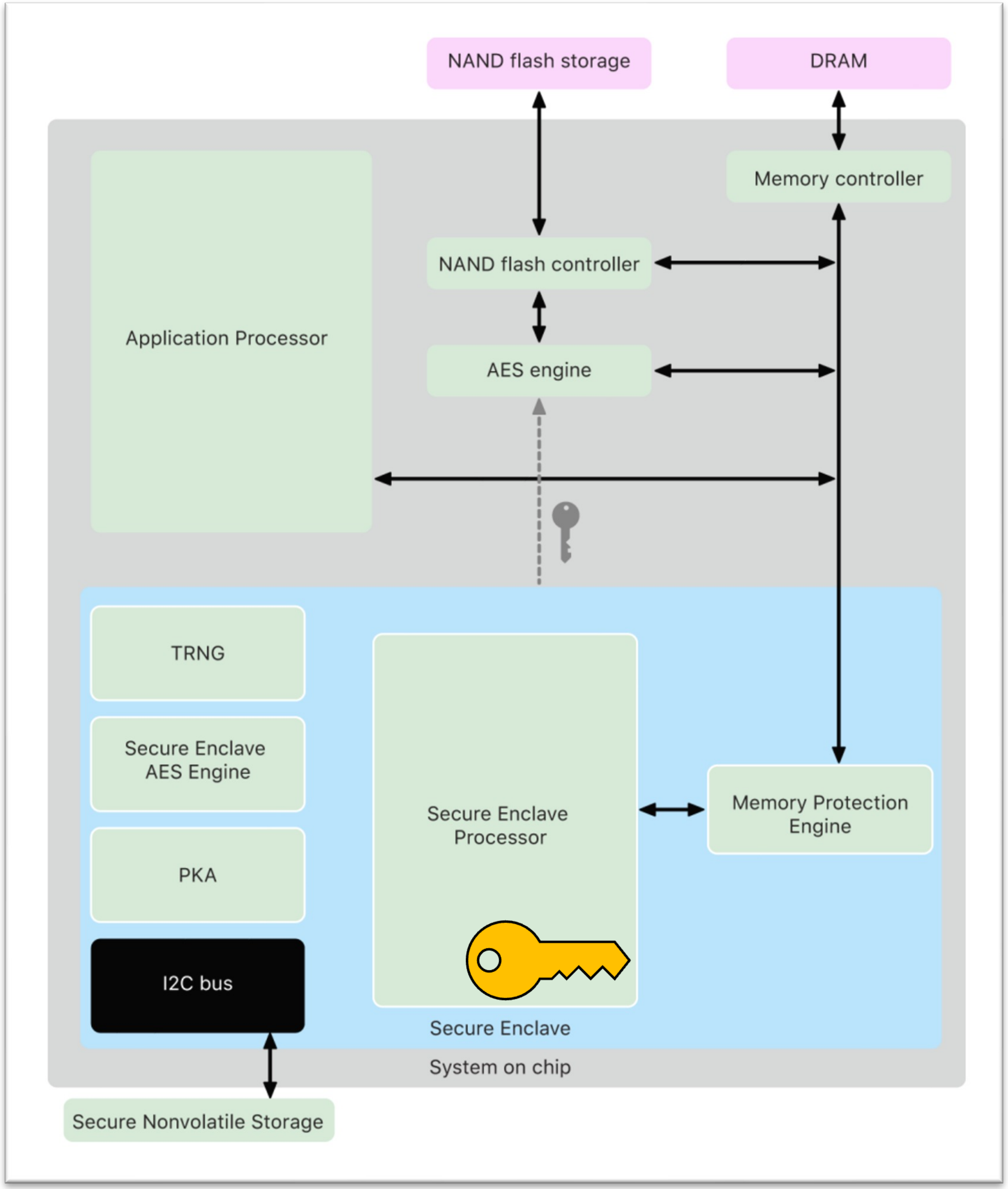
Apple Secure Enclave



Crypto Keys

The Secure Enclave includes a unique ID (**UID**) root cryptographic key.

- Unique to each device
- Randomly generated
- Fused into the SoC at manufacturing time
- Not visible outside the device



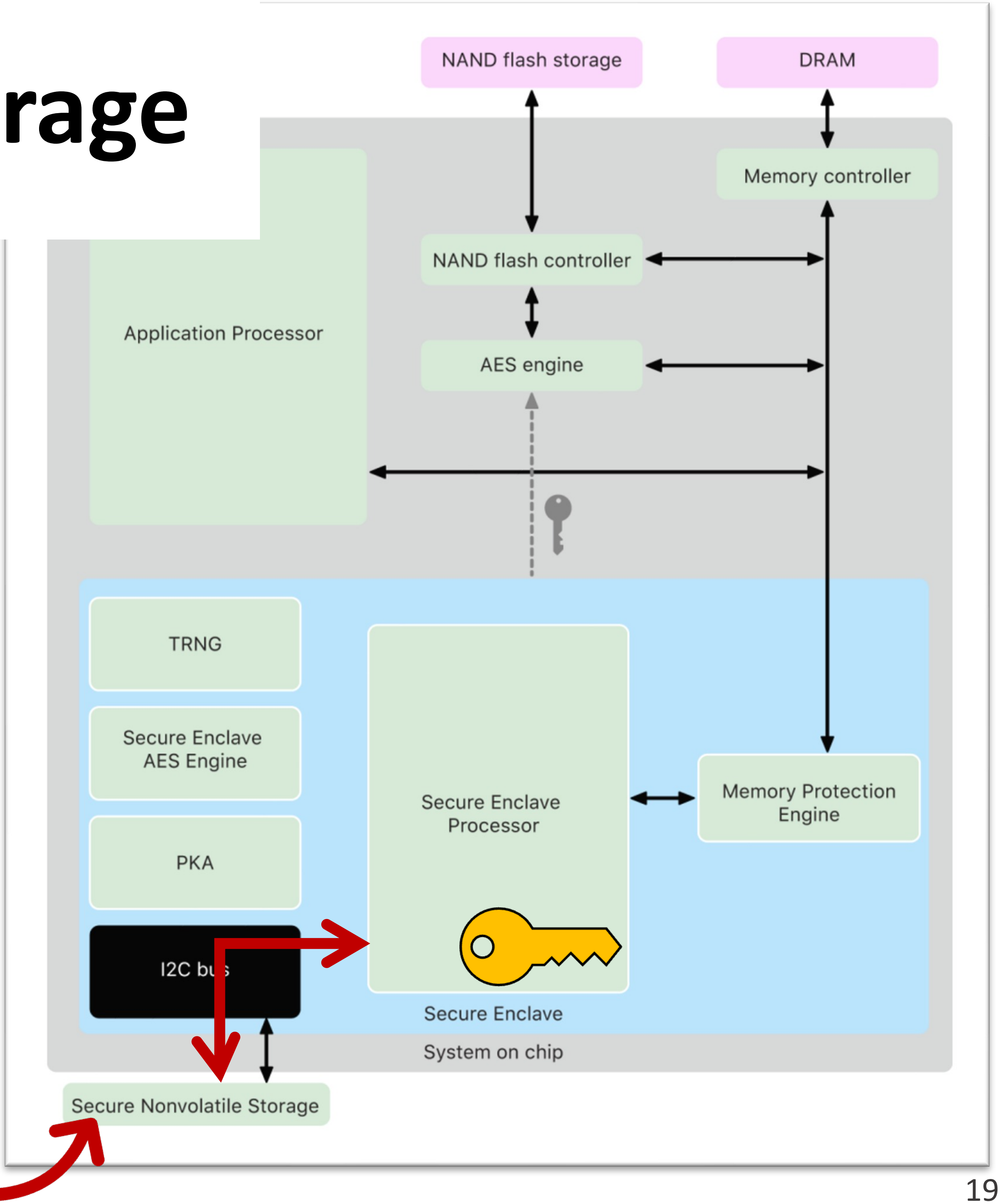
Secure Non-volatile Storage

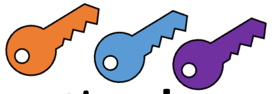
For easy to use: short passcode. But weaker security?

`Passcode + UID -> passcode entropy`

Brute-force has to be performed on the device under attack (not create a copy of the software and brute-force in parallel)

- Escalating time delays
- Erase data when exceeding attempt count

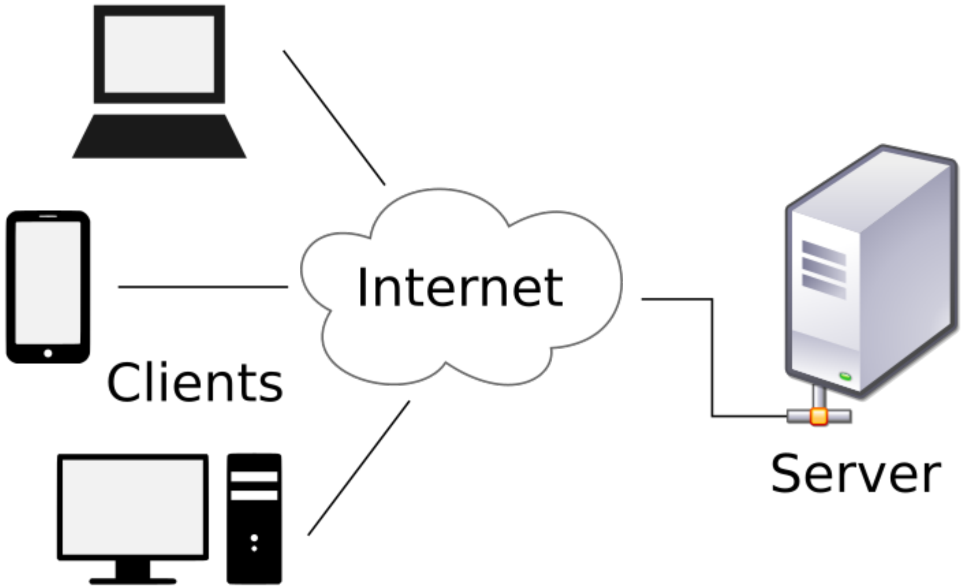




All user data encryption **keys**

Security Contexts #3

Hardware establishes root of trust.



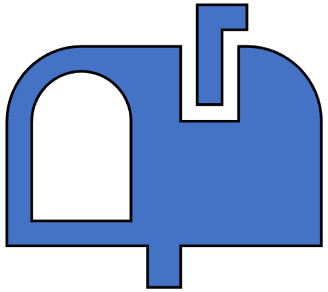
a) An end-user wants to trust a remote server, e.g., bank server.

b) A remote server wants to trust an end-user, e.g., when joining a company's highly-secure network.

c) Lost device, rootkits? Are you sure you are running your trusted OS?

Asymmetric Cryptography (e.g., RSA)

- A pair of keys:
  - Private key ($K_{private}$ – kept as secret)
  - Public key (K_{public} – safe to release publicly)



Mail box is public;
Box key is private

- Computation:
  - `Encrypt(plaintext,` K_{public}`) = ciphertext`
  - `Decrypt(ciphertext,` $K_{private}$`) = plaintext`

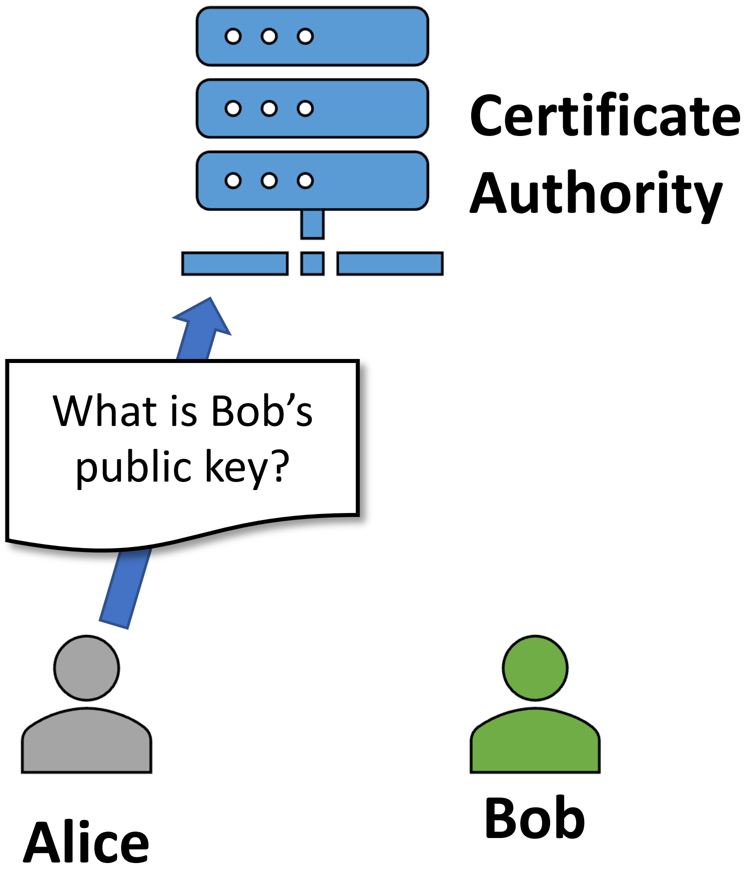
- Computationally more expensive, so usually use asymmetric cryptography to negotiate a shared key (e.g., DKE key exchange), then use symmetric cryptography

- How to announce and obtain the public key?

Public Key Infrastructures (PKIs)

- Analogy: public key is like a government-issued ID, need to be validated by an authority.

- Bob has a private key $\mathbf{K_{private}}$ and wants to claim he corresponds to a public key $\mathbf{K_{public}}$

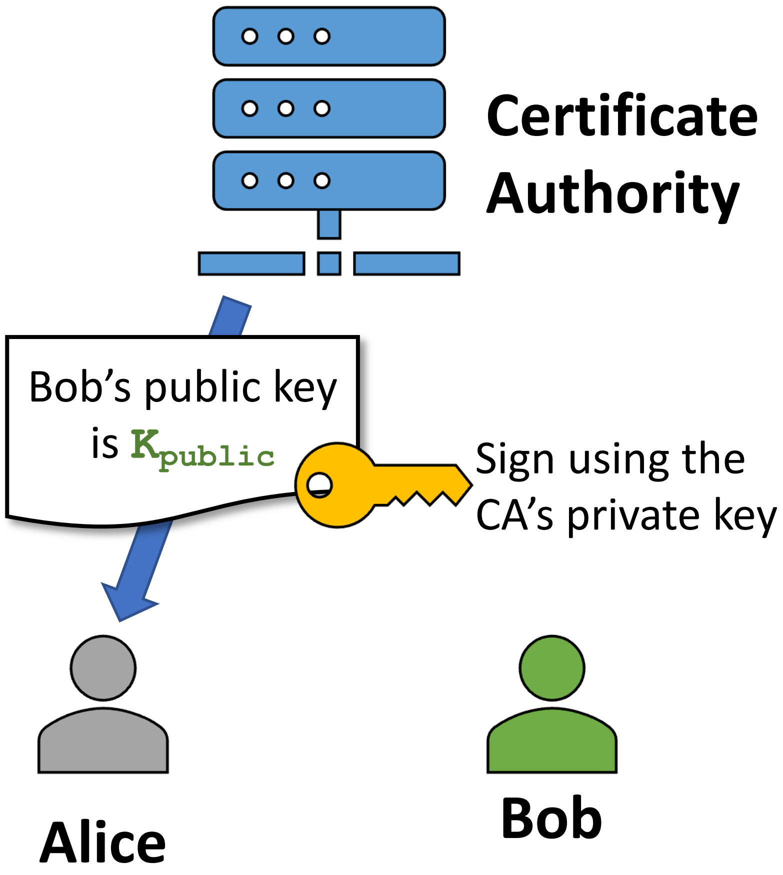


Public Key Infrastructures (PKIs)

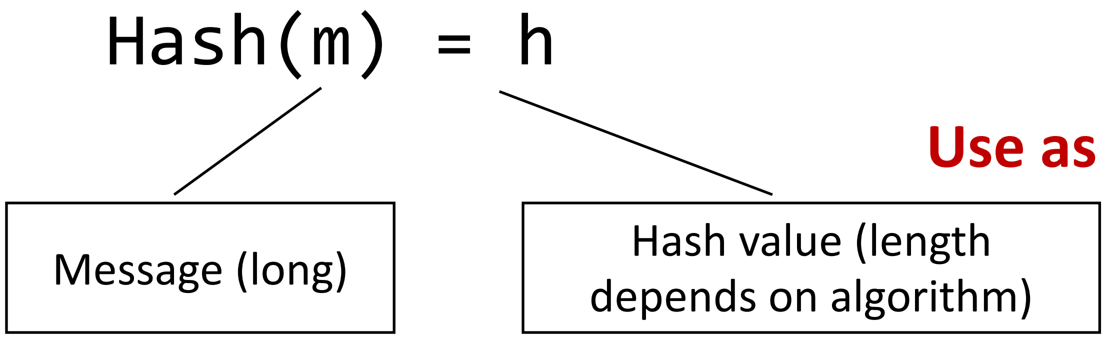
- Analogy: public key is like a government-issued ID, need to be validated by an authority.

- Bob has a private key $\mathbf{K_{private}}$ and wants to claim he corresponds to a public key $\mathbf{K_{public}}$

- Establish a chain of trust
- **Real-world use cases**: identify website, identify hardware chips/processors



Integrity (MAC/Signature)



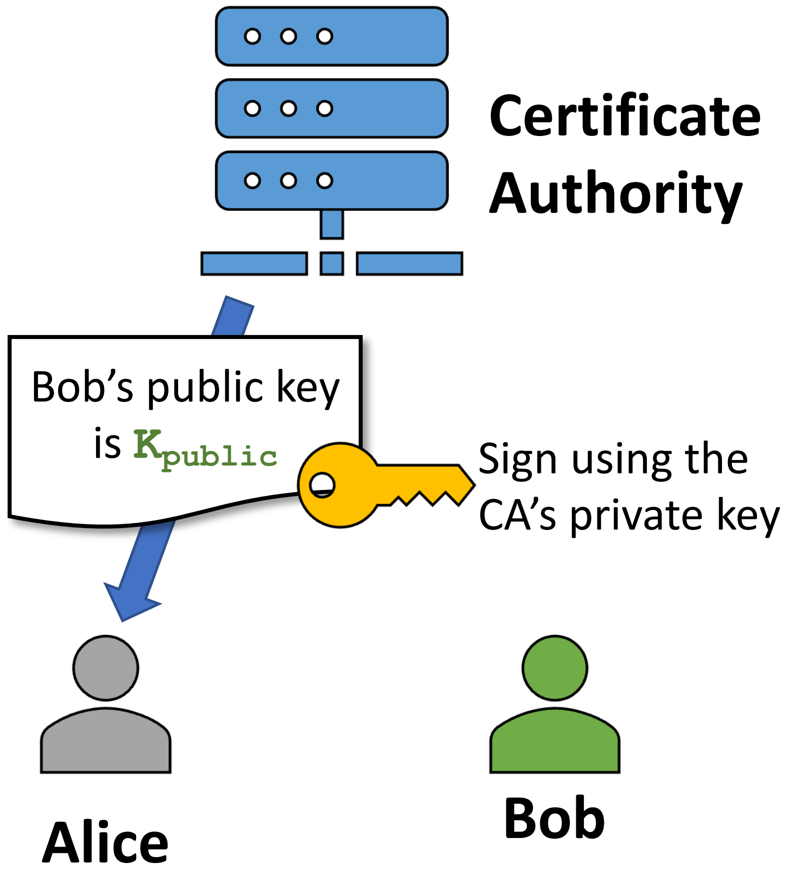
Use as fingerprints

- One-way hash
  - Practically infeasible to invert, and difficult to find collision
- Avalanche effect
  - “Bob Smith got an A+ in ELE386 in Spring 2005”→ `01eace851b72386c46d`
  - “Bob Smith got an B+ in ELE386 in Spring 2005”→ `936f8991c111f2cefaw`
- When message is long
  - Divide message into blocks, and keep extending the hash by adding previous hash

Integrity + Crypto

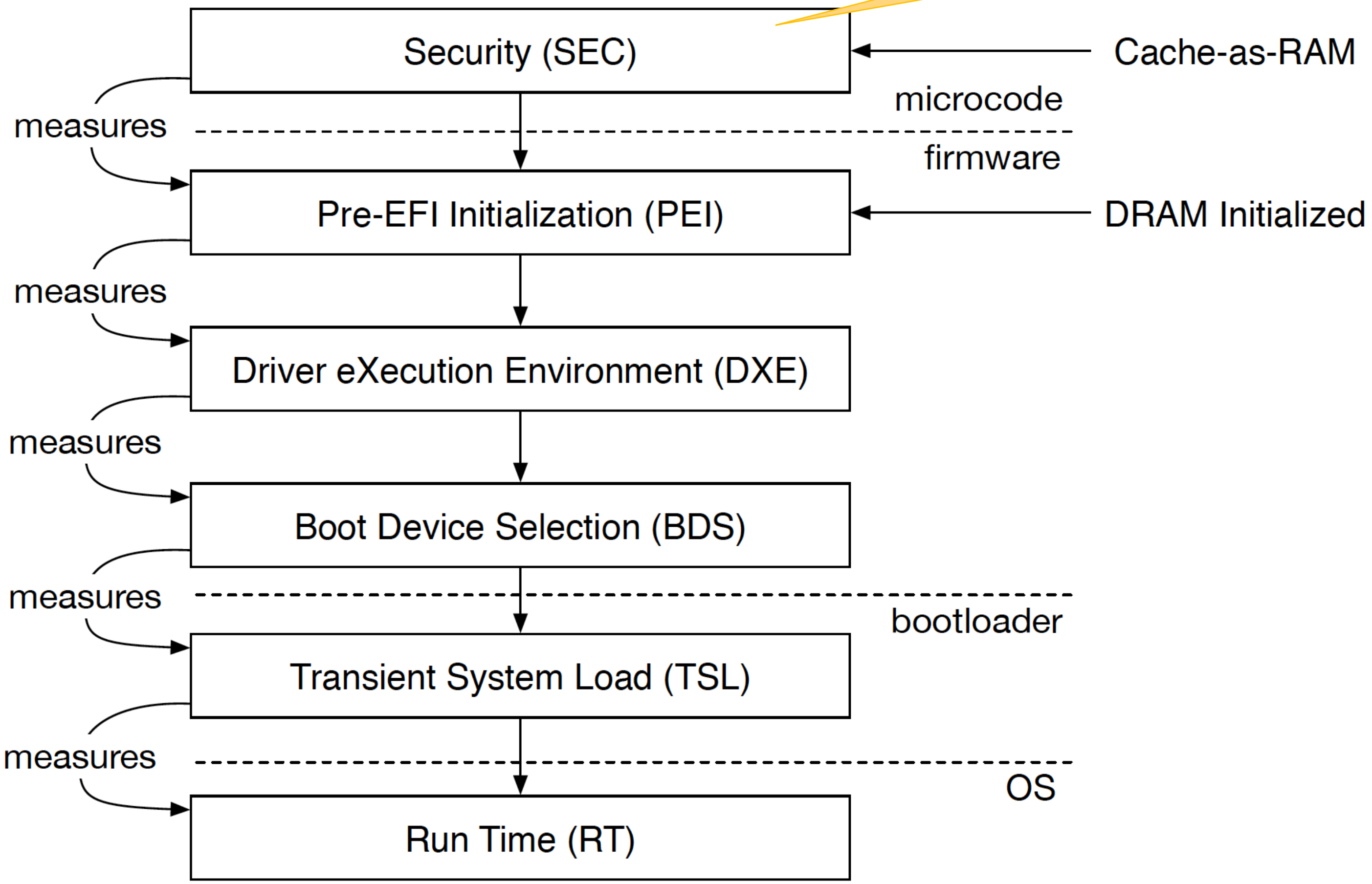
- Using symmetric crypto:
  - `hash = SHA(message)`
  - `HMAC = enc(hash, key)`

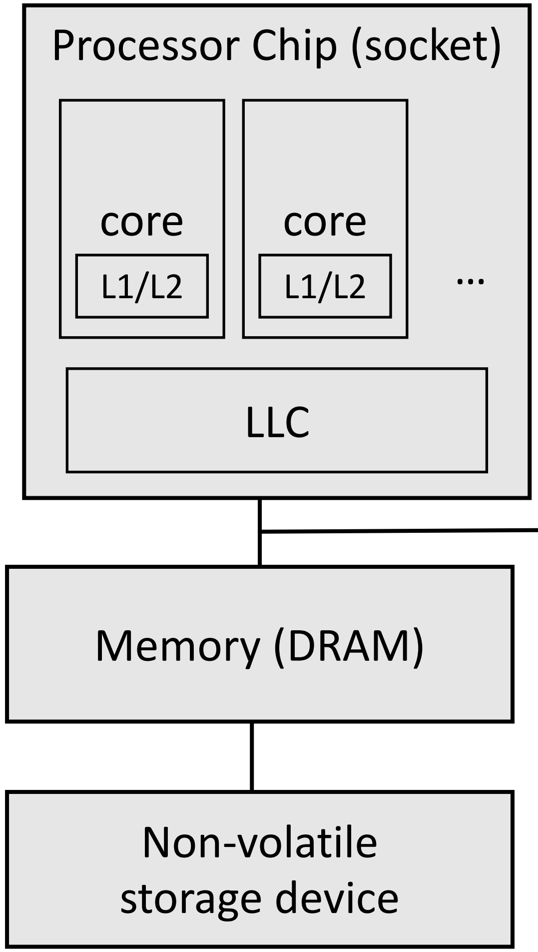
- Using asymmetric crypto:
  - Sign: `sig = dec(hash,` $K_{private}$`)`
  - Verify:
    - `hash'= SHA(message)`
    - `sig = enc(hash',` K_{public}`)`

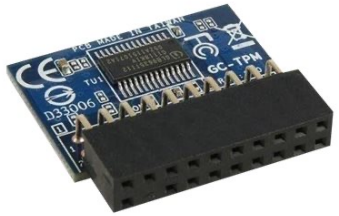


Boot Process (UEFI)

Root of trust

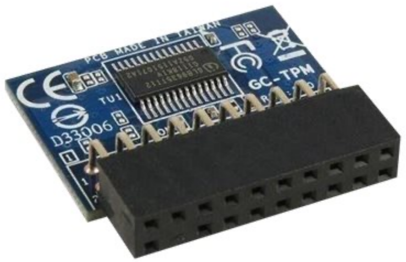


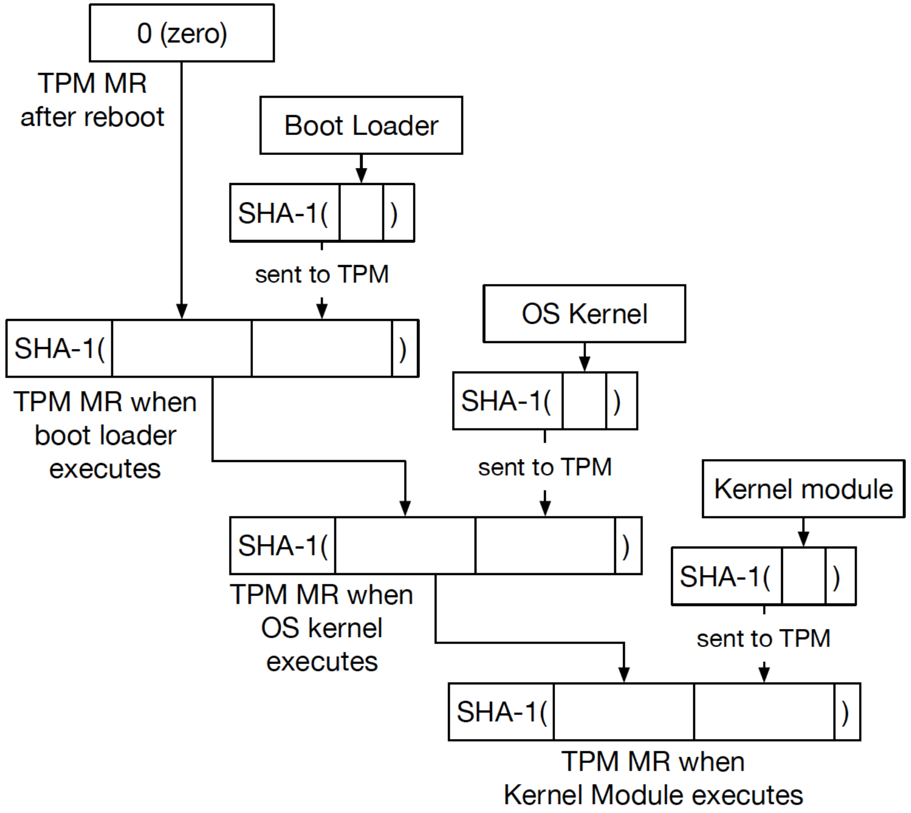


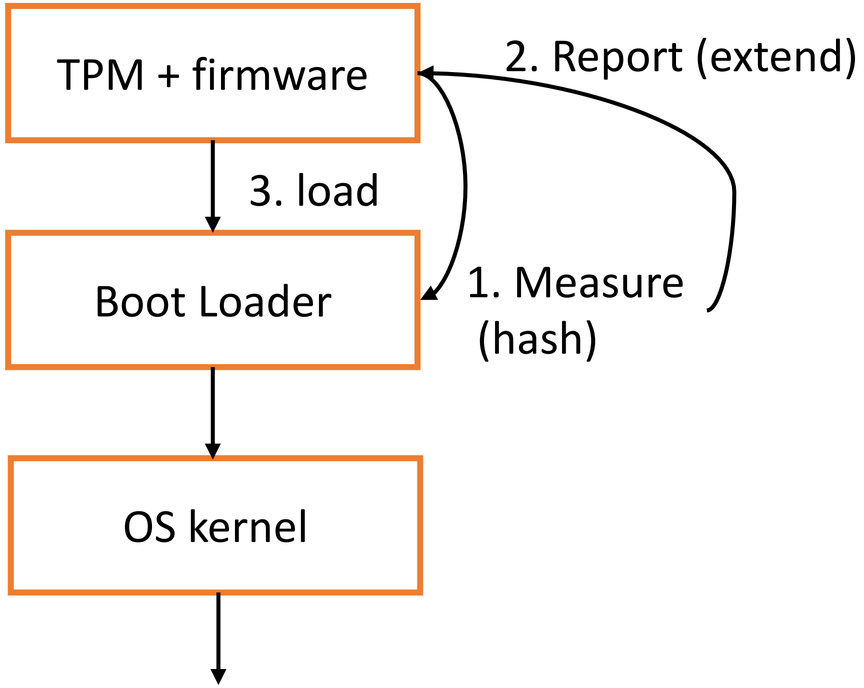


How to perform the measurement?

Secure Boot using TPM





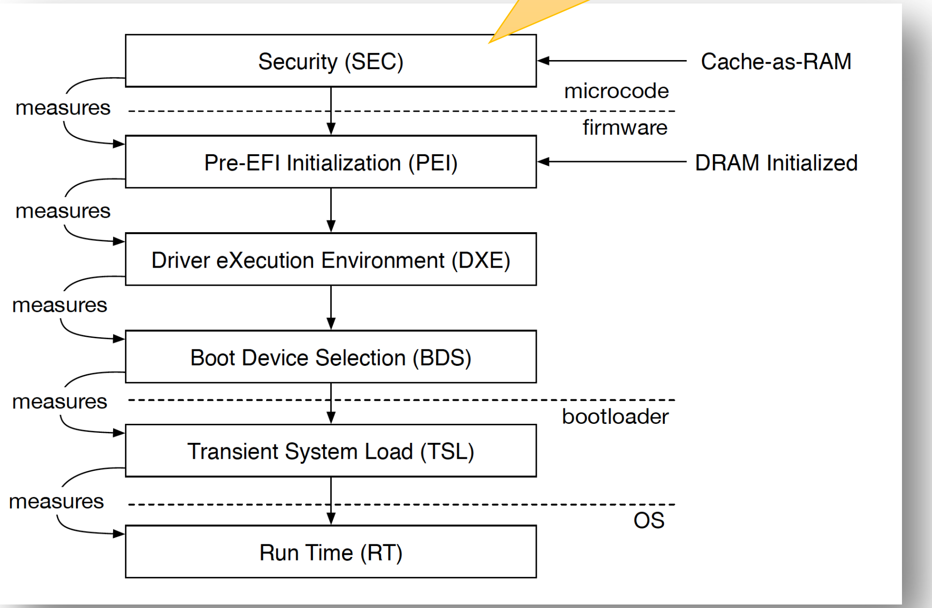


Each step, TPM compares to expected values locally or submitted to a remote attestor.

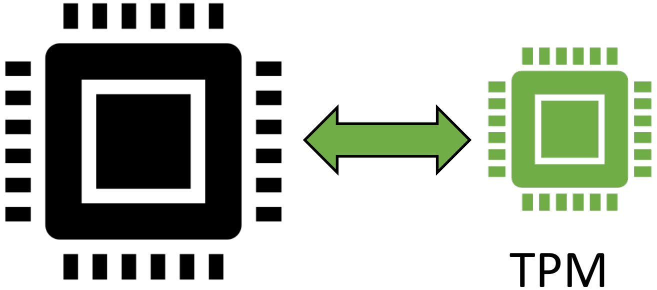
PCR: platform configuration register

Security Problems of Using TPM

Root of trust



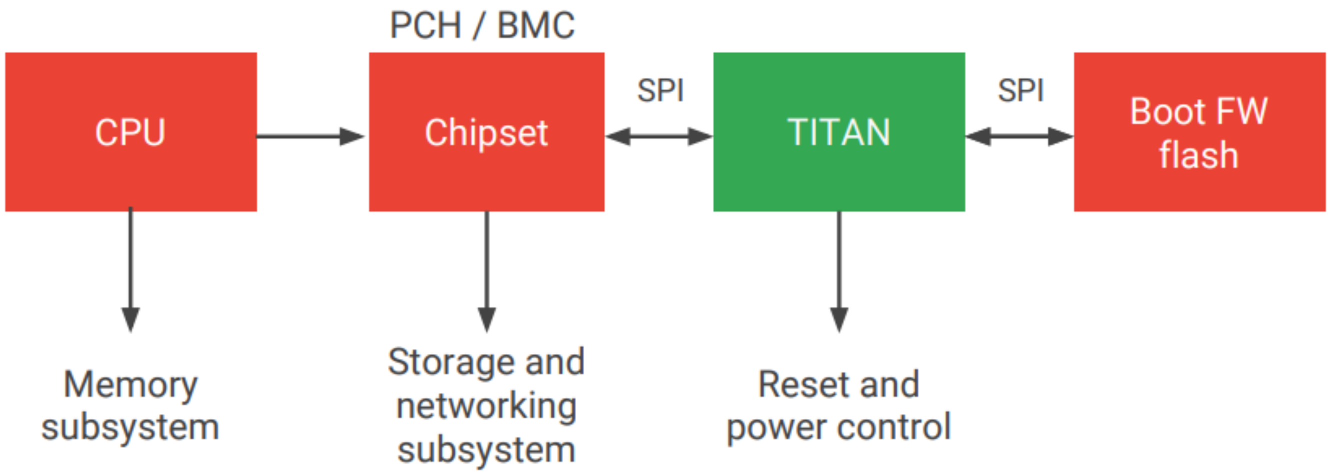
- Assume the first-stage bootloader is securely embedded in motherboard
- Not easy to use with frequent software/kernel update
- Time to check, time to use
- TPM Reset attacks
  - exploiting software vulnerabilities and using software to report false hash values

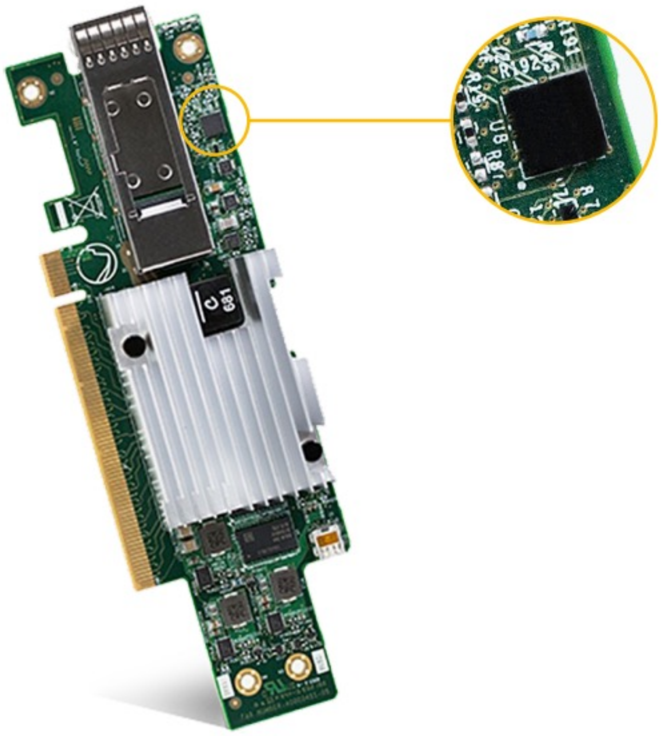


CPU TPM

Han et al. A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping. Usenix Security'18 Wojtczuk et al. Attacking Intel TXT® via SINIT code execution hijacking. 2011

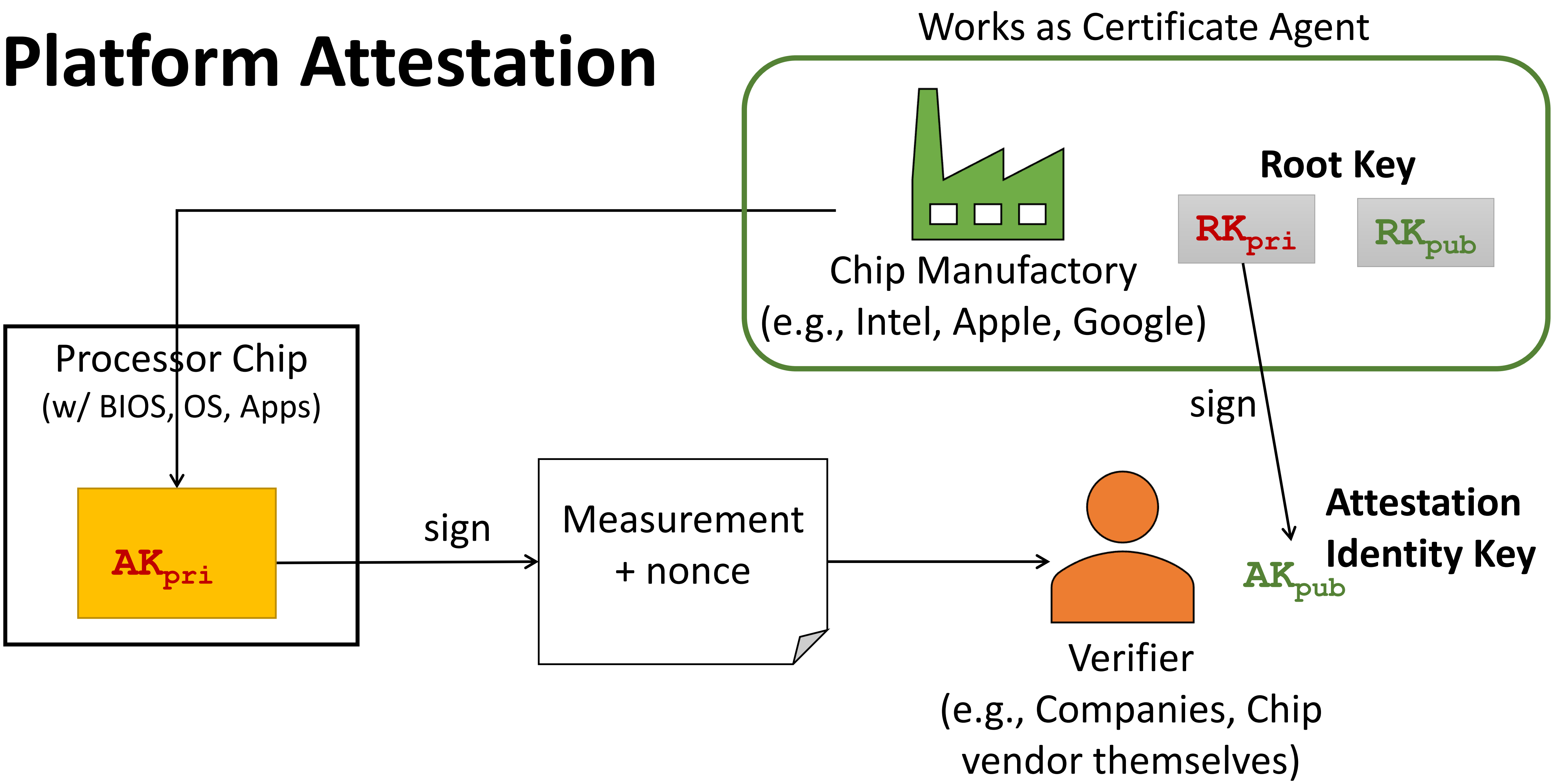
Open-source Choice: Google Titan



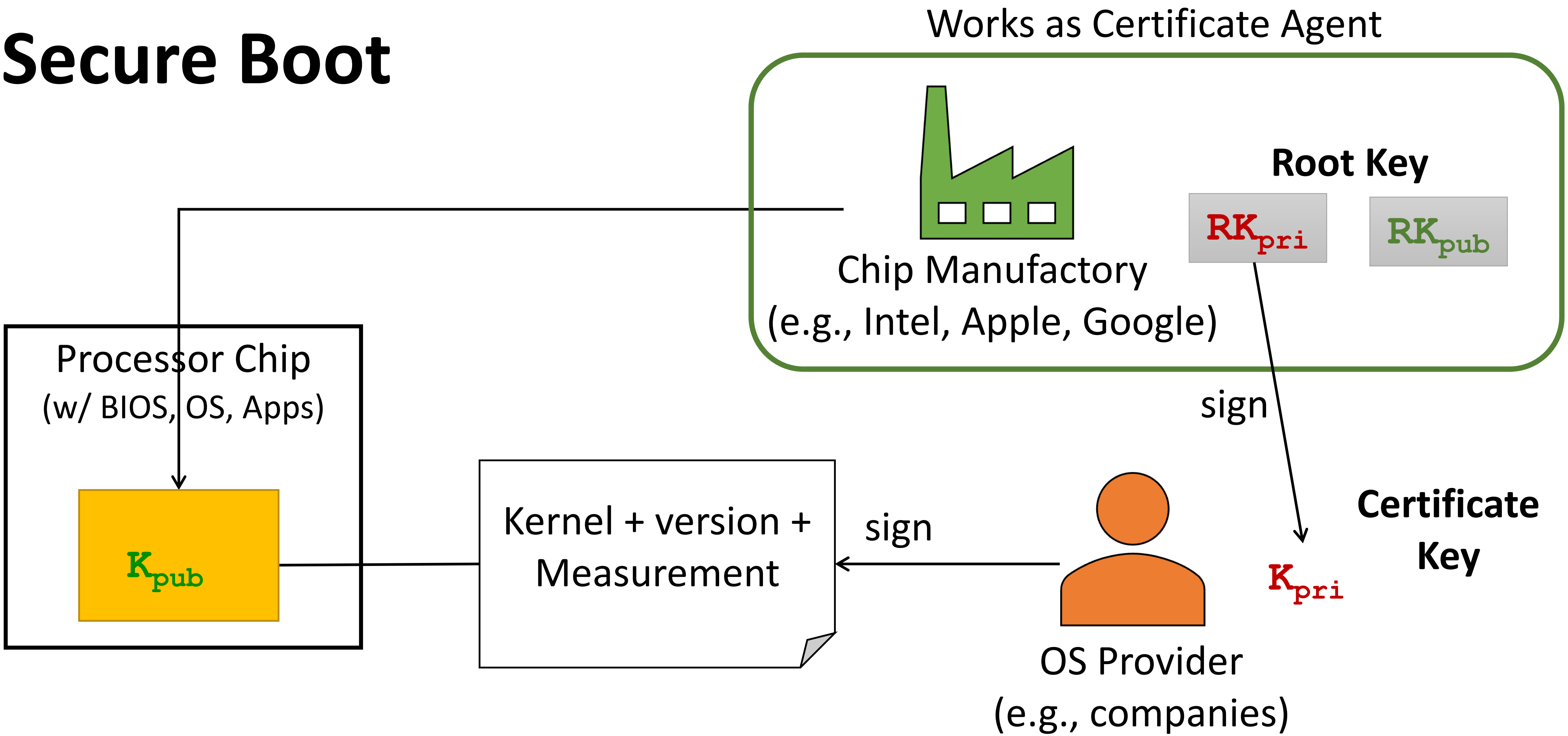


from https://www.hotchips.org/hc30/1conf/1.14_Google_Titan_GoogleFinalTitanHotChips2018.pdf

Platform Attestation



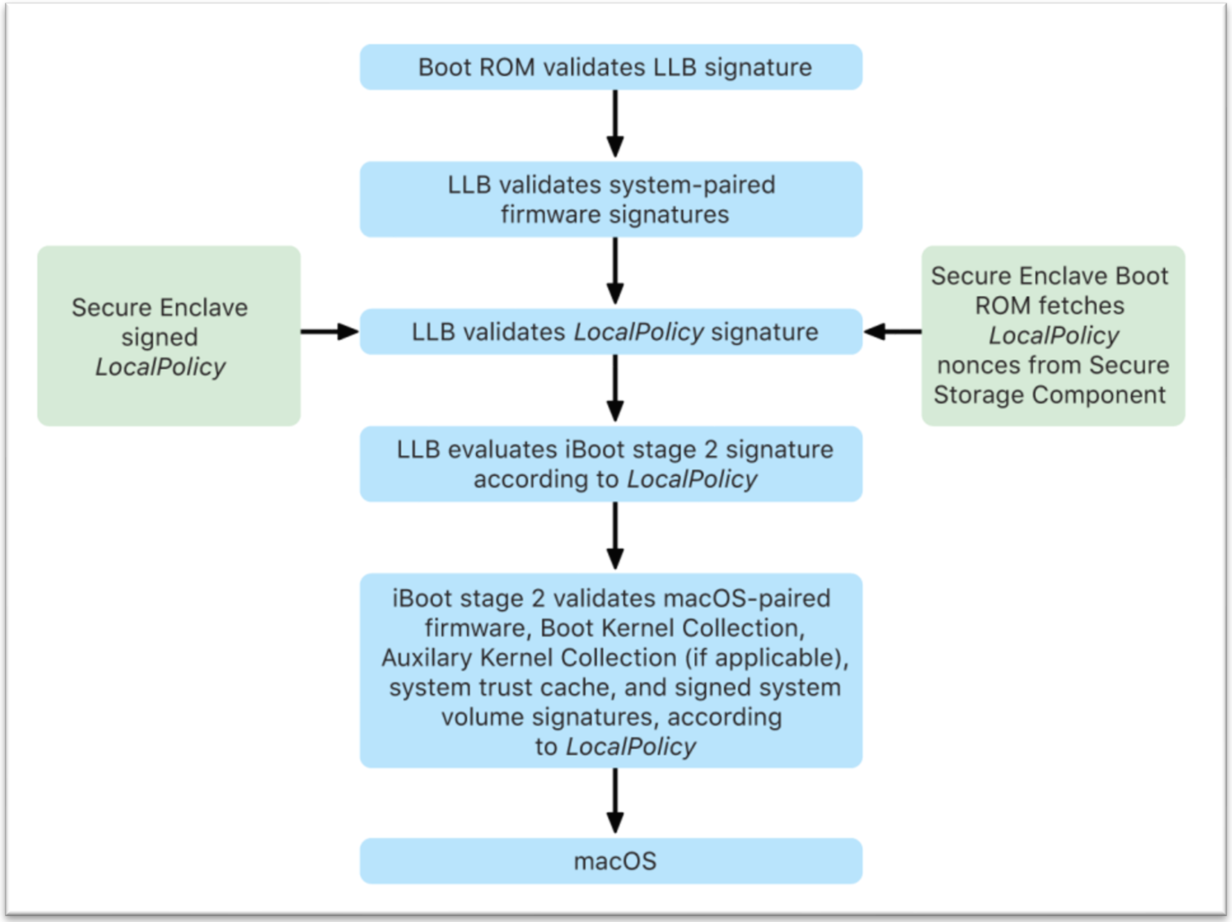
Secure Boot


Works as Certificate Agent
Chip Manufactory
(e.g., Intel, Apple, Google)
Root Key
RK_{pri}
RK_{pub}
sign
Processor Chip
(w/ BIOS, OS, Apps)
K_{pub}
Kernel + version +
Measurement
sign
OS Provider
(e.g., companies)
K_{pri}
Certificate
Key

Secure Boot

Similar to TPM but with more constraints

- Each step is signed by Apple to prevent loading non-Apple systems
- Verify more components, including operating system, kernel extensions, etc.
- Keep track of version number to prevent rolling back to older/vulnerable versions



Summary

What Can Hardware Security Modules Offer?

- Physical isolation
- Bind data and applications with the hardware device
- Establish root of trust
- More efficient

Challenges: software support. Programmability.

Next:
Physical Attacks

(with many demos 🔨🎩🤩✨)