Comp 790-184: Hardware Security and Side-Channels

Lecture 5: Hardware Security Modules

March 5, 2024

Andrew Kwong

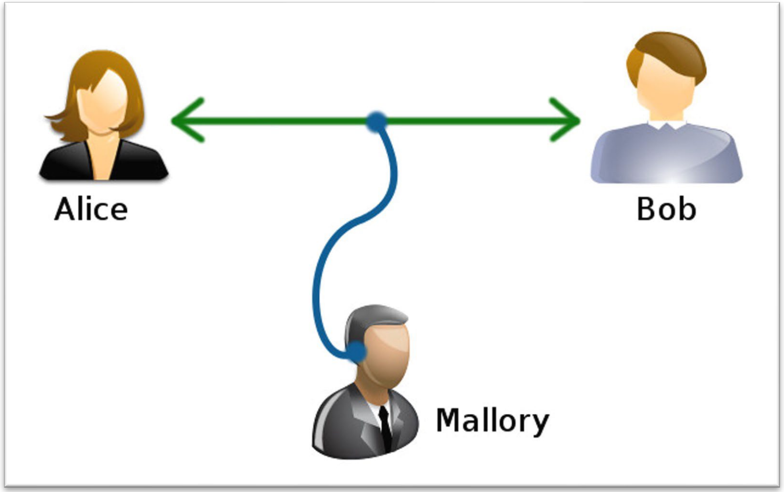
THE UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Outline

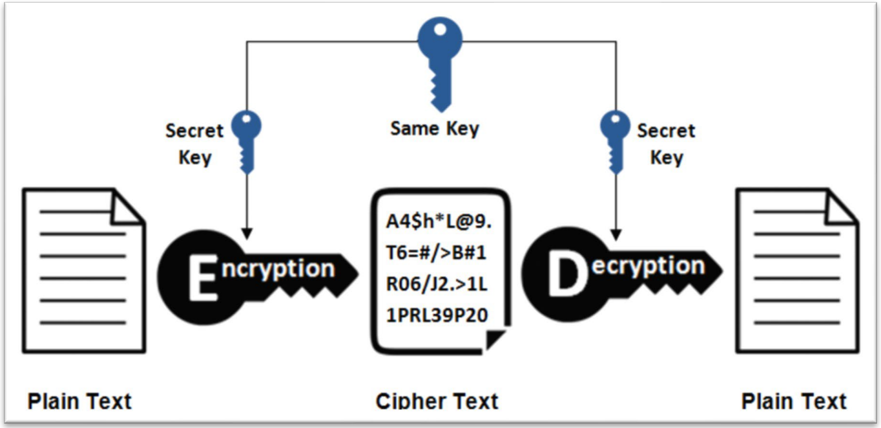
- Crypto background
  - crypto is great, but real-world security also needs hardware support
- Design considerations and tradeoffs when designing hardware security modules

Security Properties and Crypto Primitives

- Confidentiality
  - Symmetric
  - Asymmetric
- Integrity
- Freshness



Symmetric Cryptography



- One-time-pad (OTP)

Encryption:
ciphertext = key ⊕ plaintext

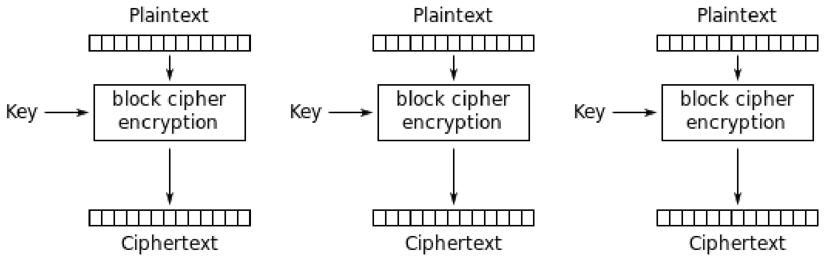
Decryption:
plaintext = key ⊕ ciphertext

How about encrypting arbitrary length message? Any problems?

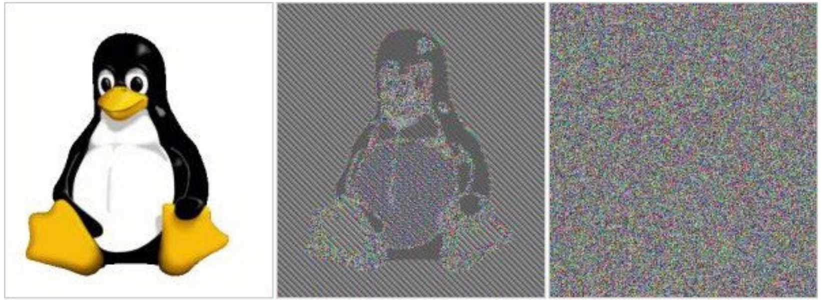
Block ciphers (e.g., DES, AES)

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

ECB IS NOT RECOMMENDED

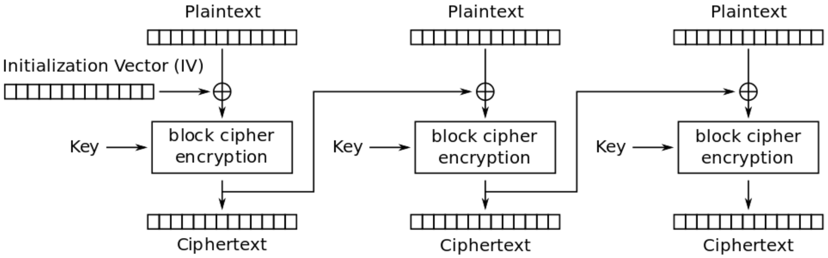


Electronic Codebook (ECB) mode encryption

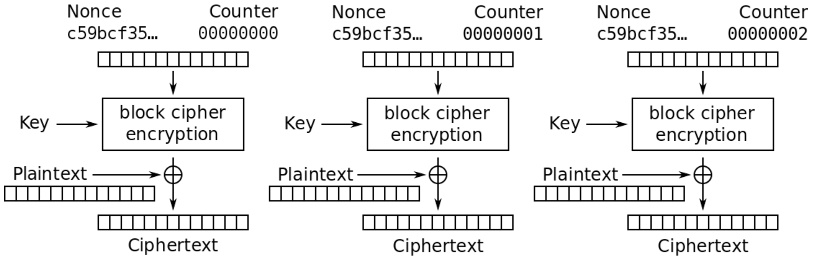


Original image | Encrypted using ECB mode | Modes other than ECB result in pseudo-randomness

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



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.

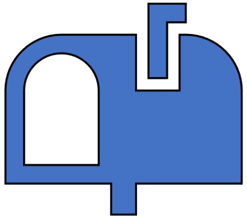
Real-world application: file/disk encryption and memory encryption.

How do we exchange the shared key between two parties?

Asymmetric Cryptography (e.g., RSA)

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

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



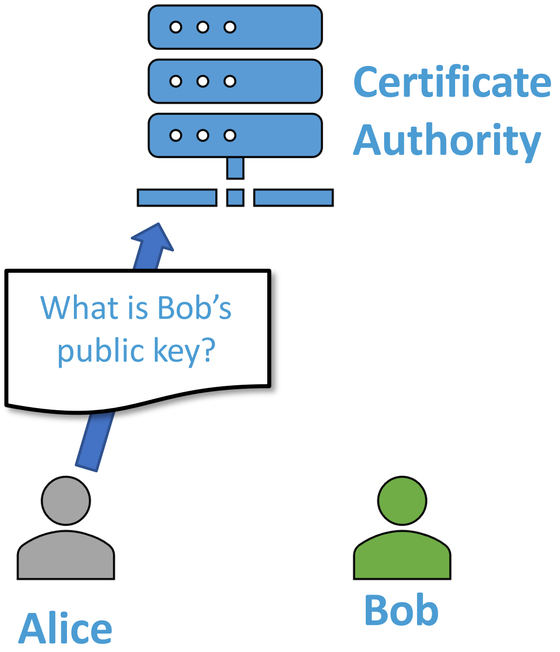
Mail box is public;
Box key is private

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

- How do we announce and obtain the public key?

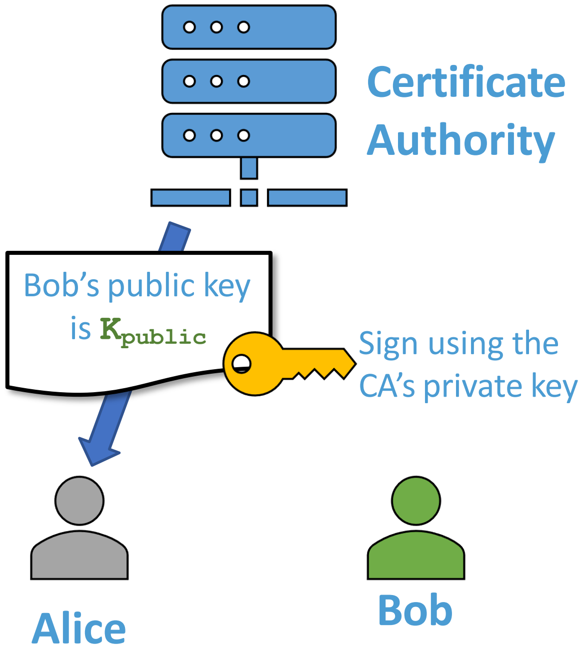
Public Key Infrastructures (PKIs)

- Bob has a private key $\mathbf{K}_{\texttt{private}}$ and wants to claim he corresponds to a public key $\mathbf{K}_{\texttt{public}}$
- Analogy: public key is like a government-issued ID, need to be validated by an authority.



Public Key Infrastructures (PKIs)

- Bob has a private key $K_{private}$ and wants to claim he corresponds to a public key K_{public}
- Analogy: public key is like a government-issued ID, need to be validated by an authority.
- Establish a chain of trust
- **Real-world use cases**: identify website, identify hardware chips/processors



Integrity (MAC/Signature)

`Hash(m) = h`

Message (long)

Hash value (length depends on algorithm)

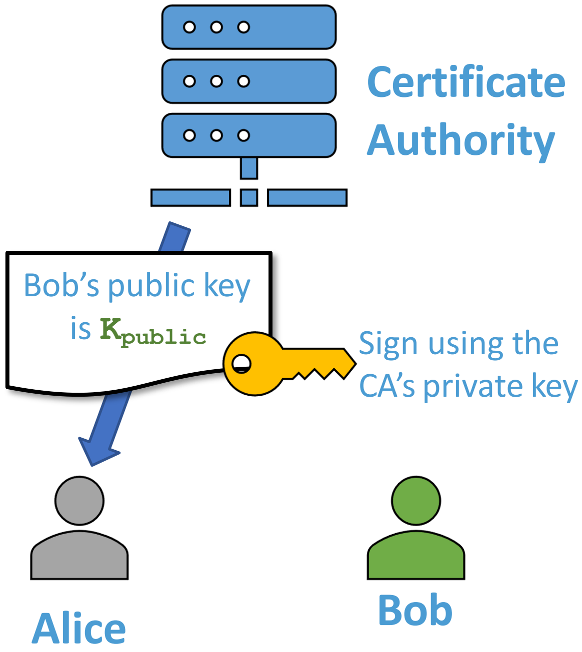
Use as fingerprints

- Hash: one-way function
  - Practically infeasible to invert, and difficult to find collision
- Avalanche effect
  - “Bob Smith got an A+ in ELE386 in Spring 2005”→ `01eace851b72386c46`
  - “Bob Smith got an B+ in ELE386 in Spring 2005”→ `936f8991c111f2cefaw`

Integrity + Crypto

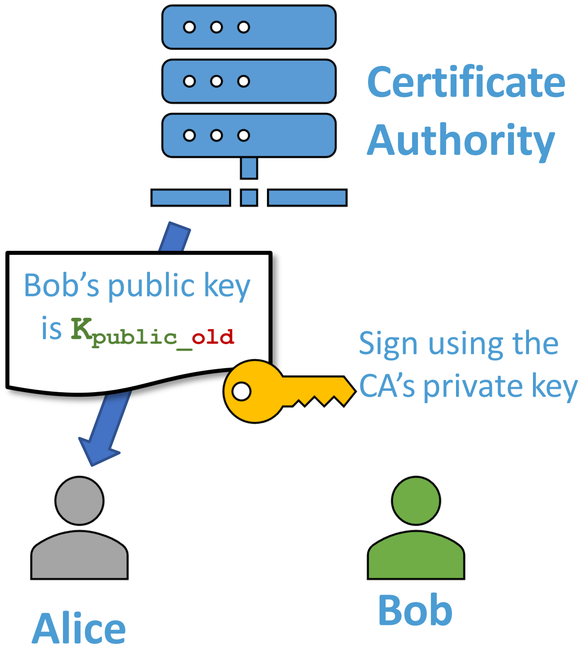
- Using symmetric crypto:
  - `hash = SHA(message)`
  - `HMAC = enc(hash, key)`

- Using asymmetric crypto:
  - Sign: $sig = sign(hash, K_{private})$
  - Verify:
    - $ver = verify(sig, K_{public}, message)$



Freshness

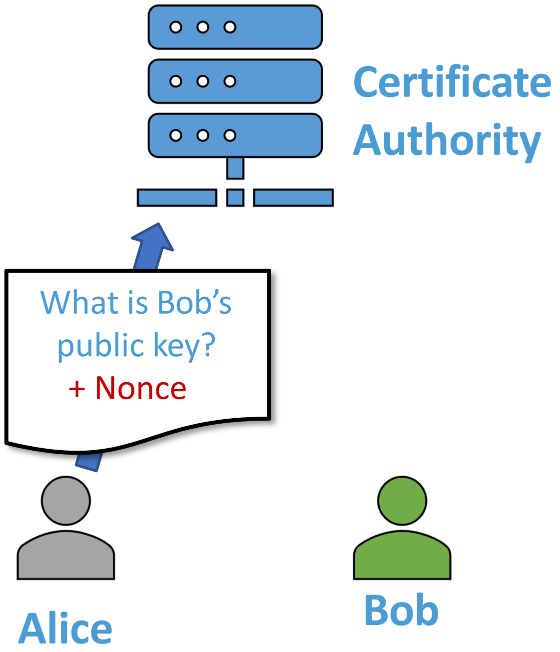
- Goal: to block replay attack



Freshness

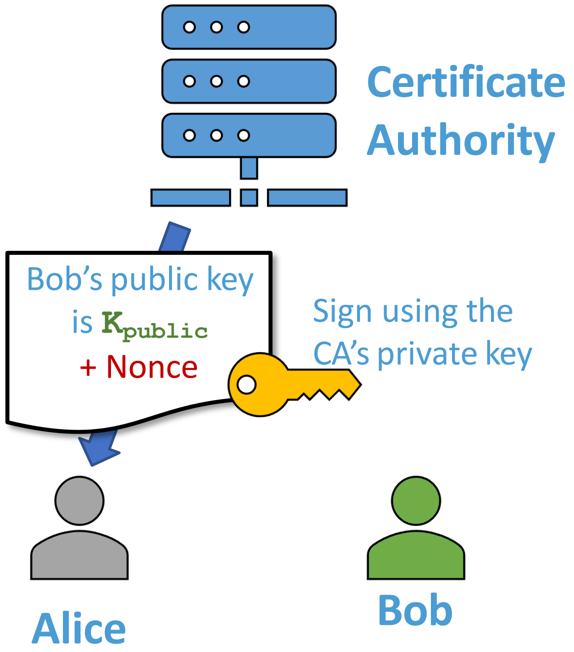
- Goal: to block replay attack

- Nonces + Integrity
  - Nonce is a one-time use random number
  - Should not reuse the same nonce



Freshness

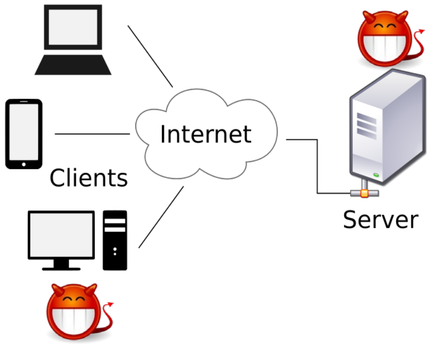
- Goal: to block replay attack
- Nounces + Integrity
  - Nonce is a one-time use random number
  - Should not reuse the same nounce
- Challenge-response



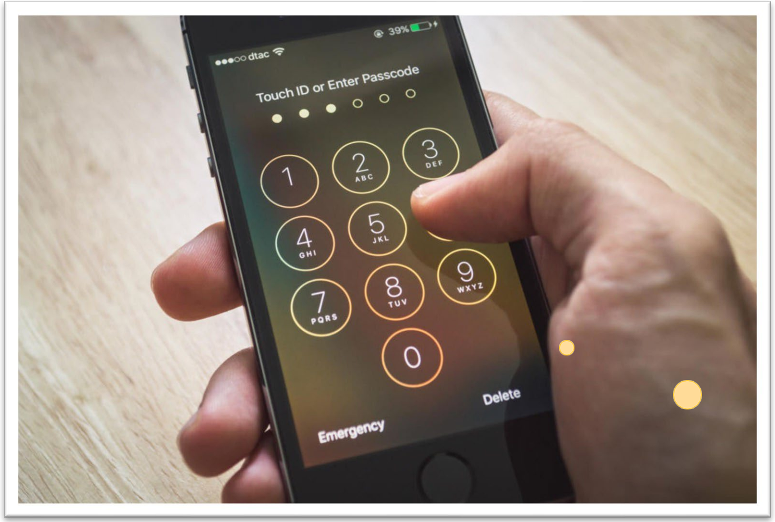
Security Contexts #1

Hardware establishes root of trust.

a) A remote server wants to trust an end-user, e.g., when joining a company's highly-secure network.
b) An end-user wants to trust a remote server, e.g., bank server
c) Lost device, rootkits?



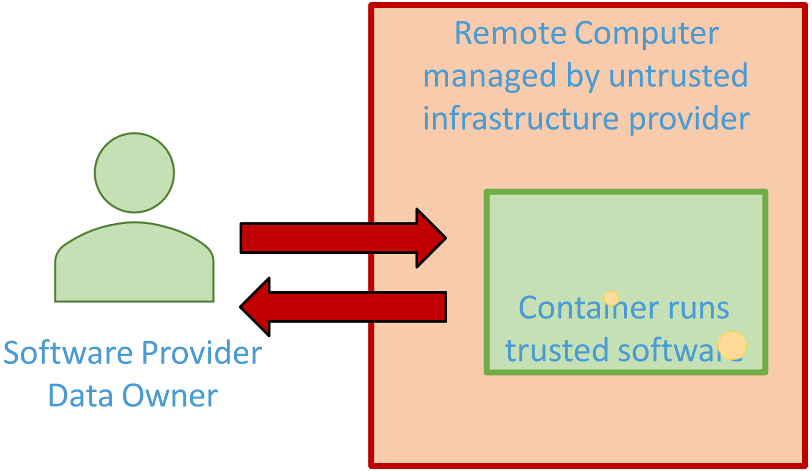
Security Contexts #2



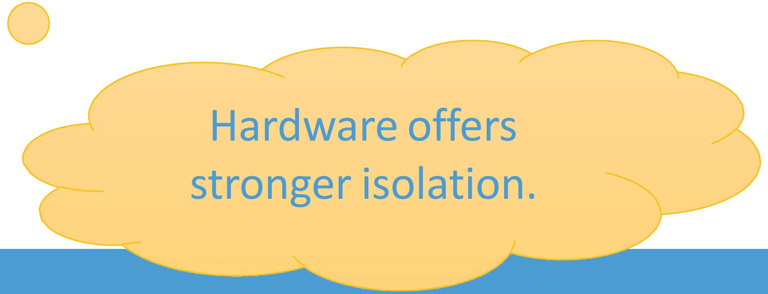
- Software piracy (copying and reselling software to gain benefits).
- Disk lost or removed, leading to confidentiality leakage.
- Data encryption with weak passwords, such as, 6-digit passcode.

Bind data/application with hardware.

Security Contexts #3



- Remote computation where the host hypervisor and OS is not trusted.



What Can Hardware Security Modules Offer?

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

Secure Processors

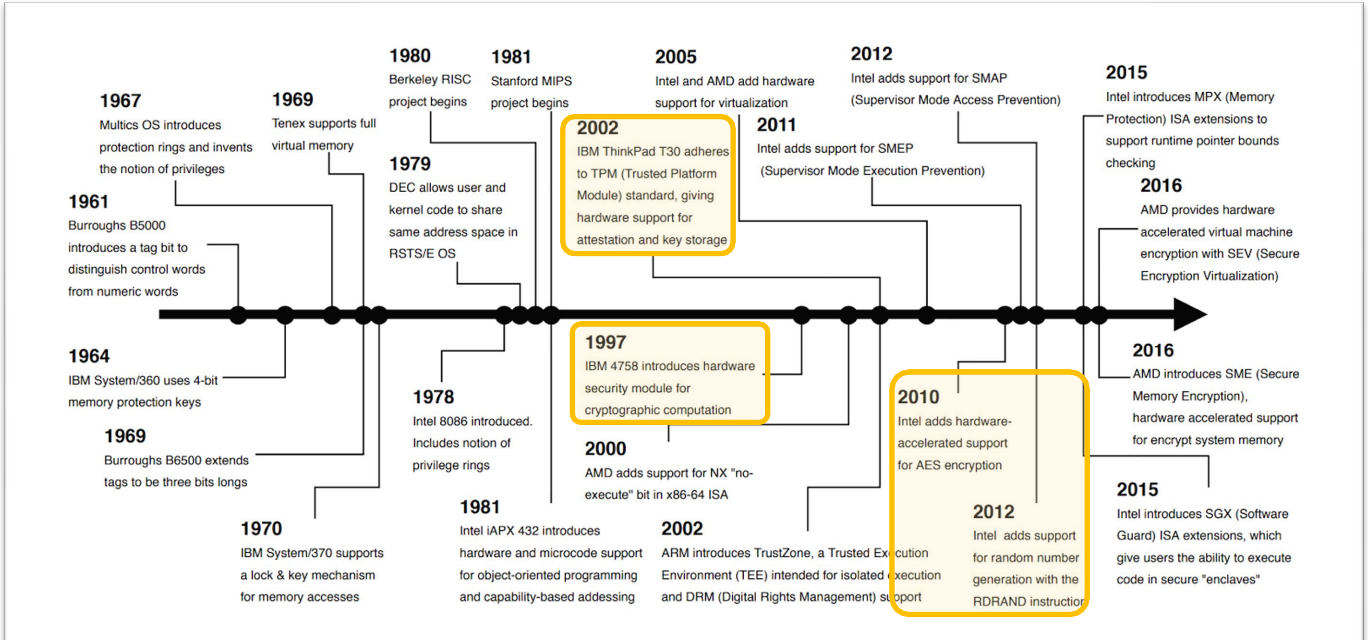
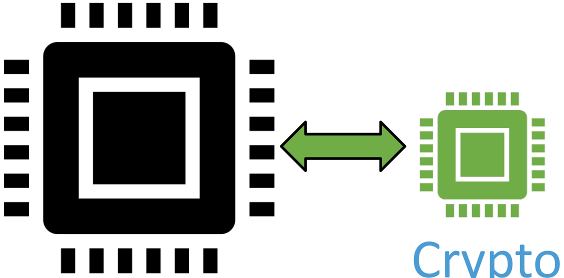


Figure 1: Sixty years of hardware support for security

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

Before IBM 4758 (1999)

- Crypto Accelerators
  - Better performance
  - Simple functionality
  - Narrow interface

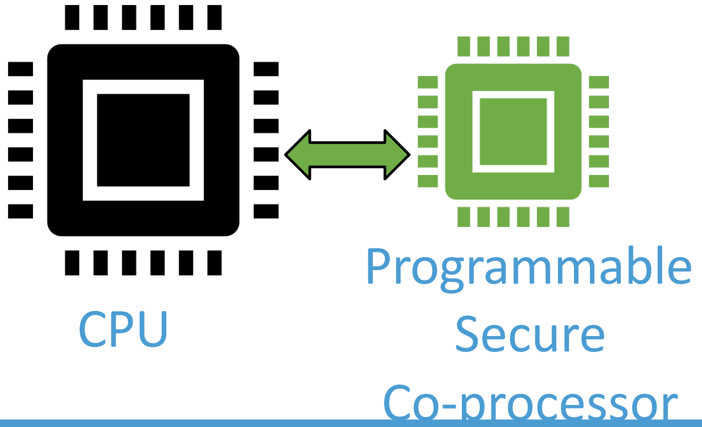


CPU

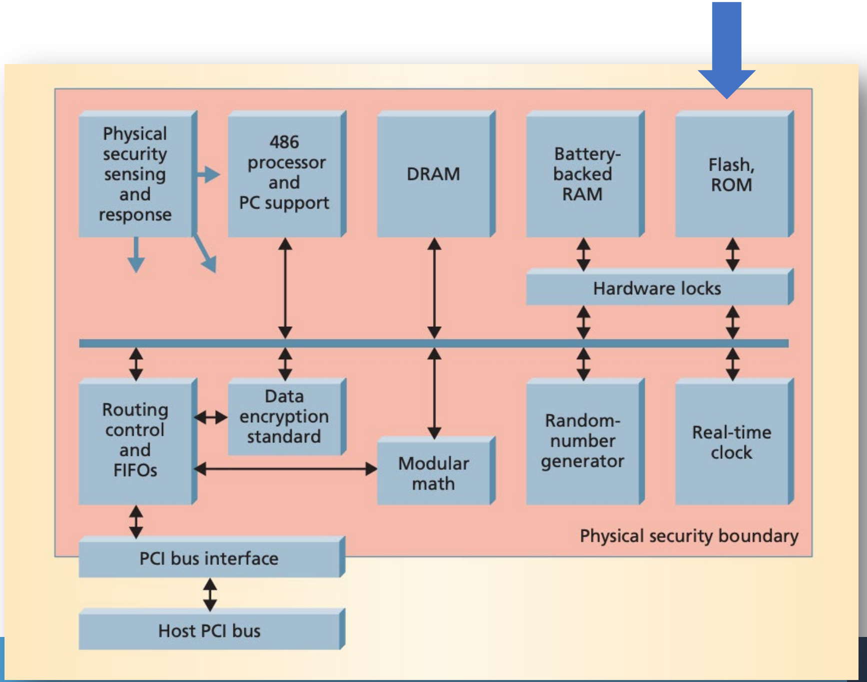
Crypto
ASIC
(AES, RSA, etc.)

IBM 4758 (1999) -- 4765 (2012)

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

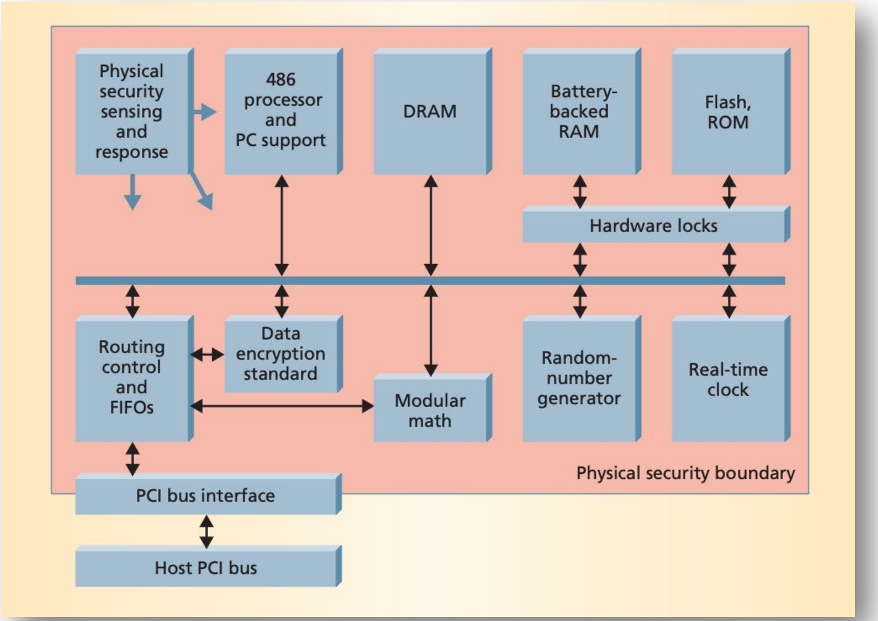


Stores the firmware and secret keys



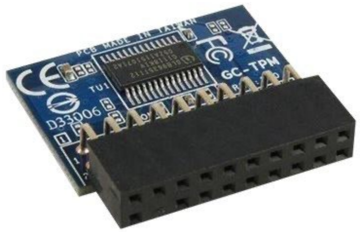
Software Layer Design and Concerns

- Software stack:
  - Application
  - OS, kernel (microkernel)
  - Loaders, firmware, etc.
- Use cases:
  - Solve music/software piracy issue
  - Run an SSL server inside to store the agreed symmetric session keys
- Problems:
  - Updating software is tricky
  - Bad programmability due to microkernel

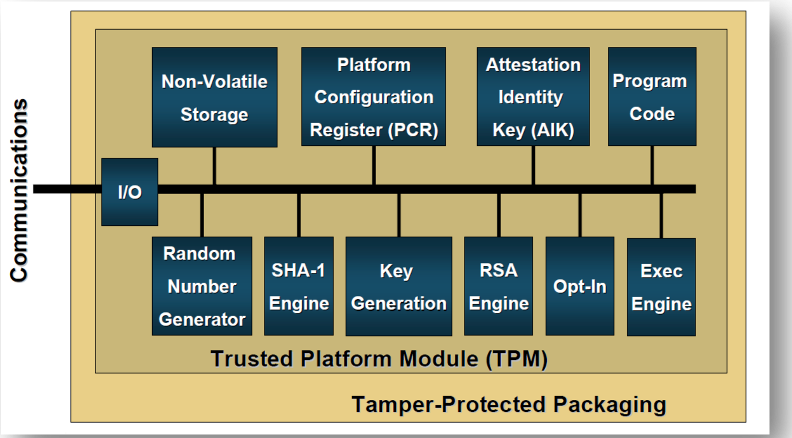


Trusted Platform Module (TPM)

- Standard LPC interface – attaches to commodity motherboards
- Weaker computation capability

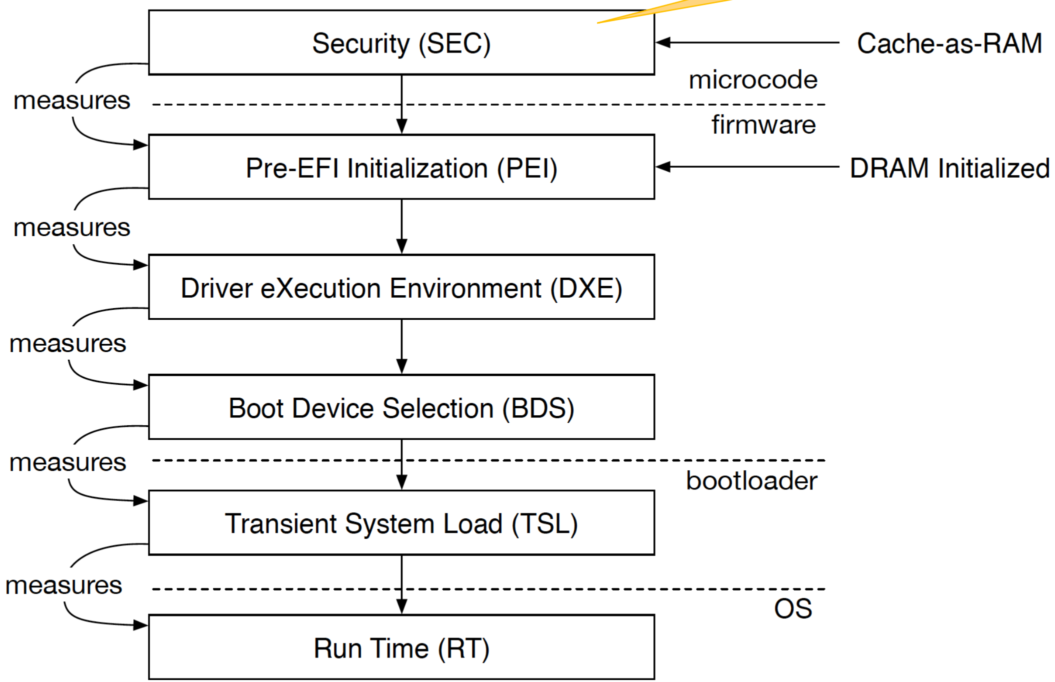


- Use cases:
  - Disk encryption and password protection (“seal”)
  - Verify platform integrity (firmware+OS)



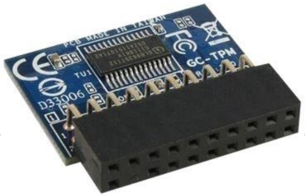
Boot Process (UEFI)

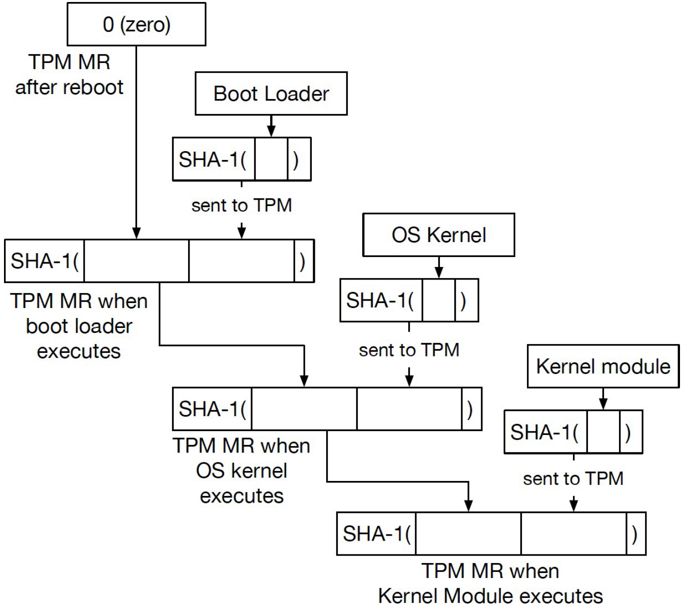
Root of trust

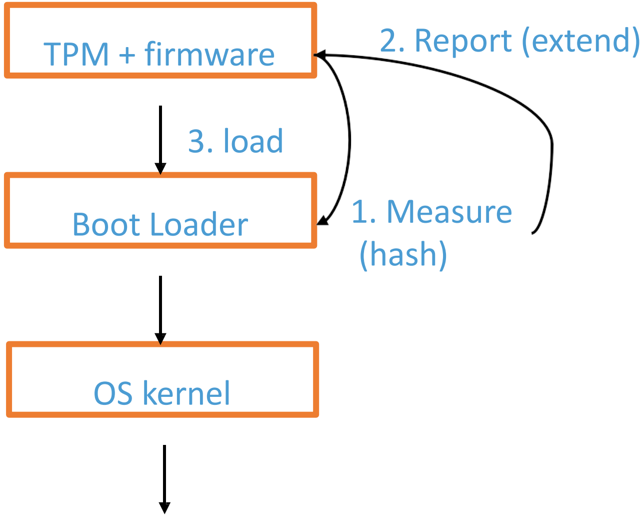


How does it perform the measurement?

Secure Boot using TPM

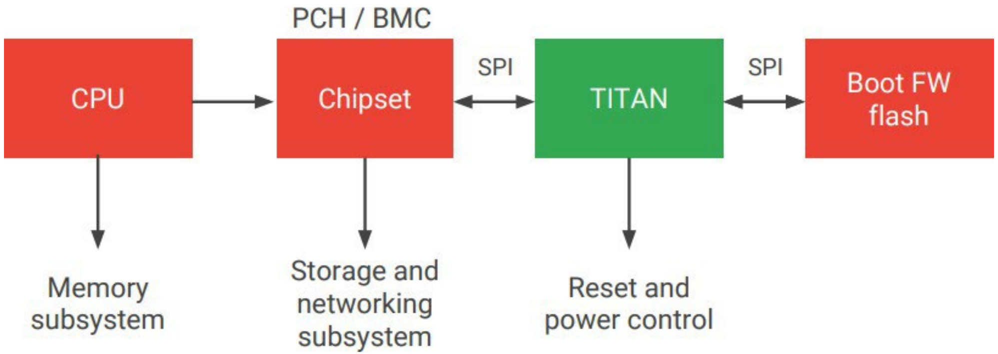


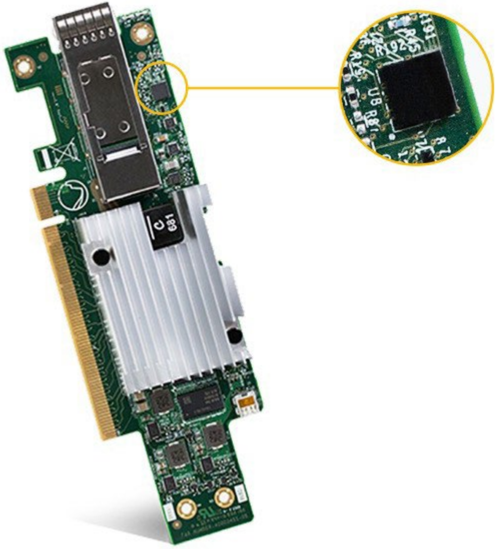




Each step, TPM compares to expected values locally or submitted to a remote attestor.

Open-source Choice: Google Titan

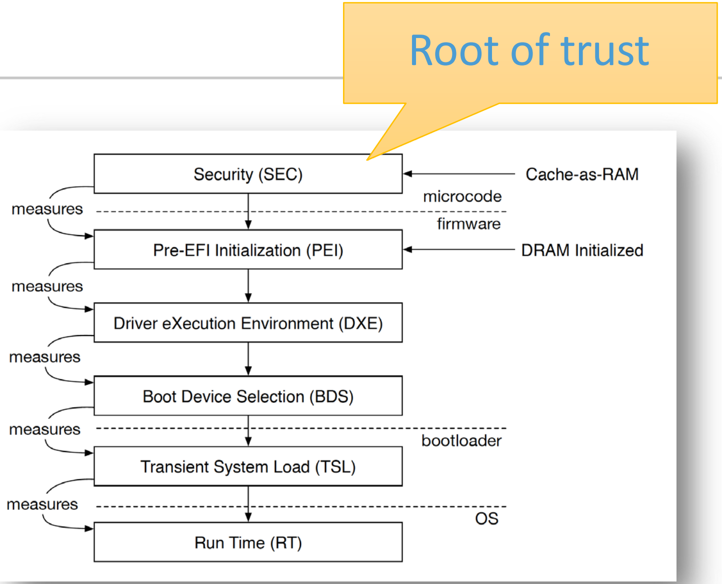


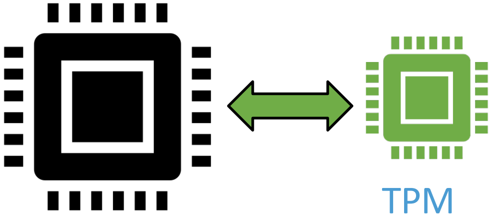


from https://www.hotchips.org/hc30/1conf/1.14_Google_Titan_GoogleFinalTitanHotChips2018.pdf

Security Problems of Using TPM

- 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



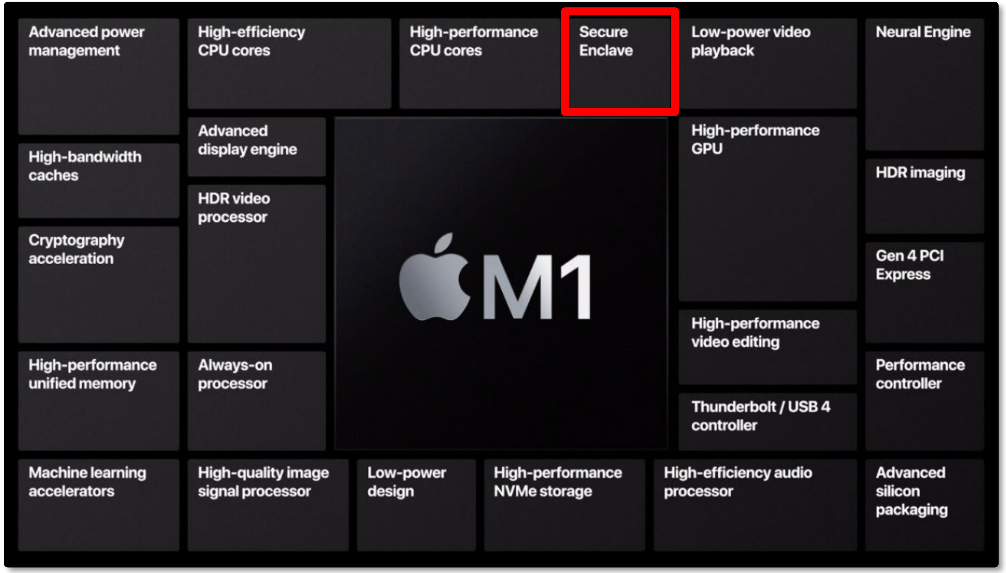


Han et al. A Bad Dream: Subverting Trusted Platform Module While You Are Sleeping. Usenix Security'18 Wojtczuk et al. Attacking Intel TXT® via SINIT code execution hijacking. 2011

Apple Secure Enclave

- Additional Goals:
  - Prevent jailbreak
  - Easy to use

- Advantage: one company controls both the hardware and the software



Isolation

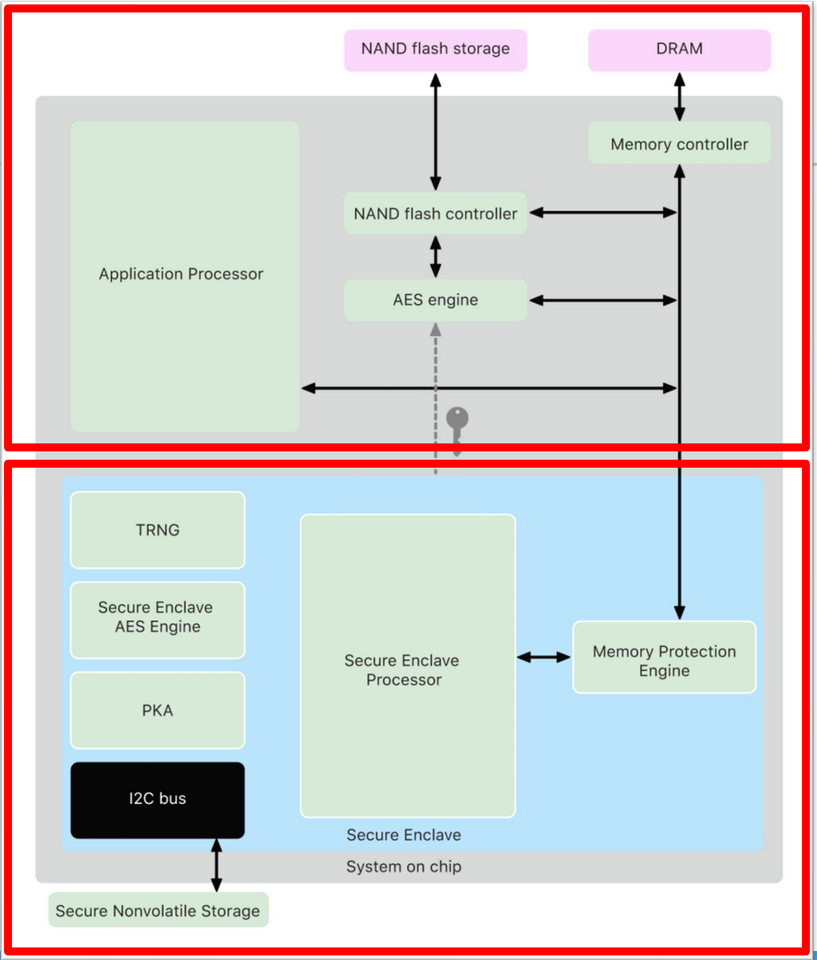
Why separate cores?

Similar to IBM 4758

- Strong isolation
- Block vulnerabilities due to software bugs and side channels

Different from IBM 4758

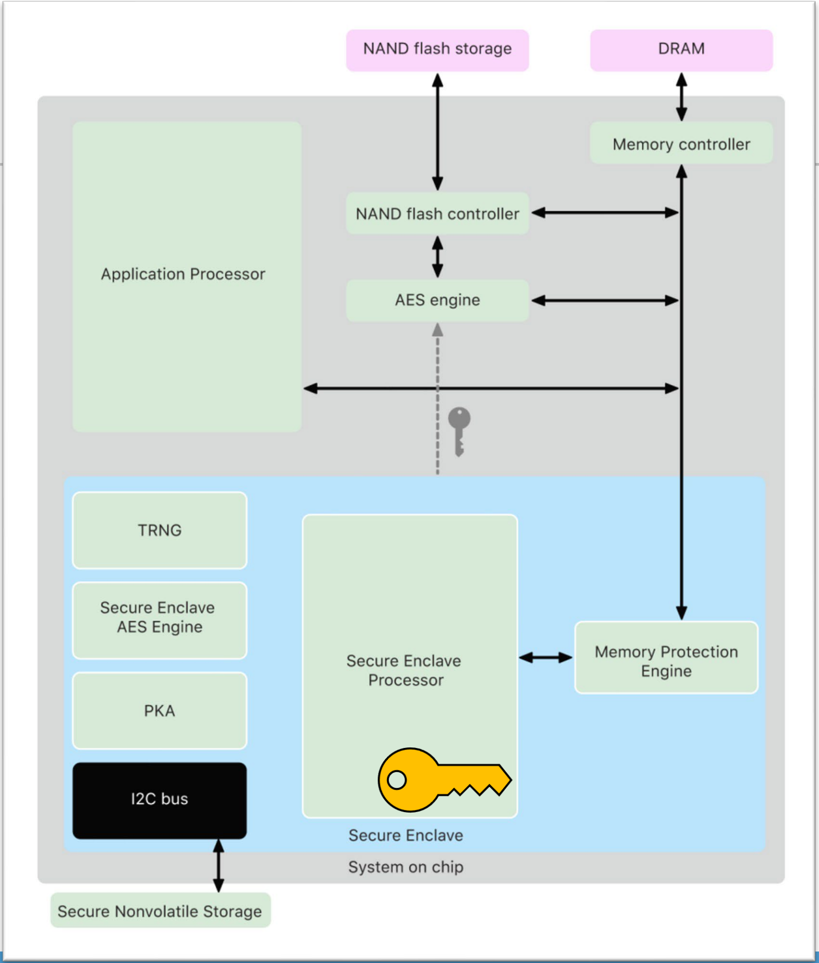
- Not general-purpose, only run secure enclave functionality



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



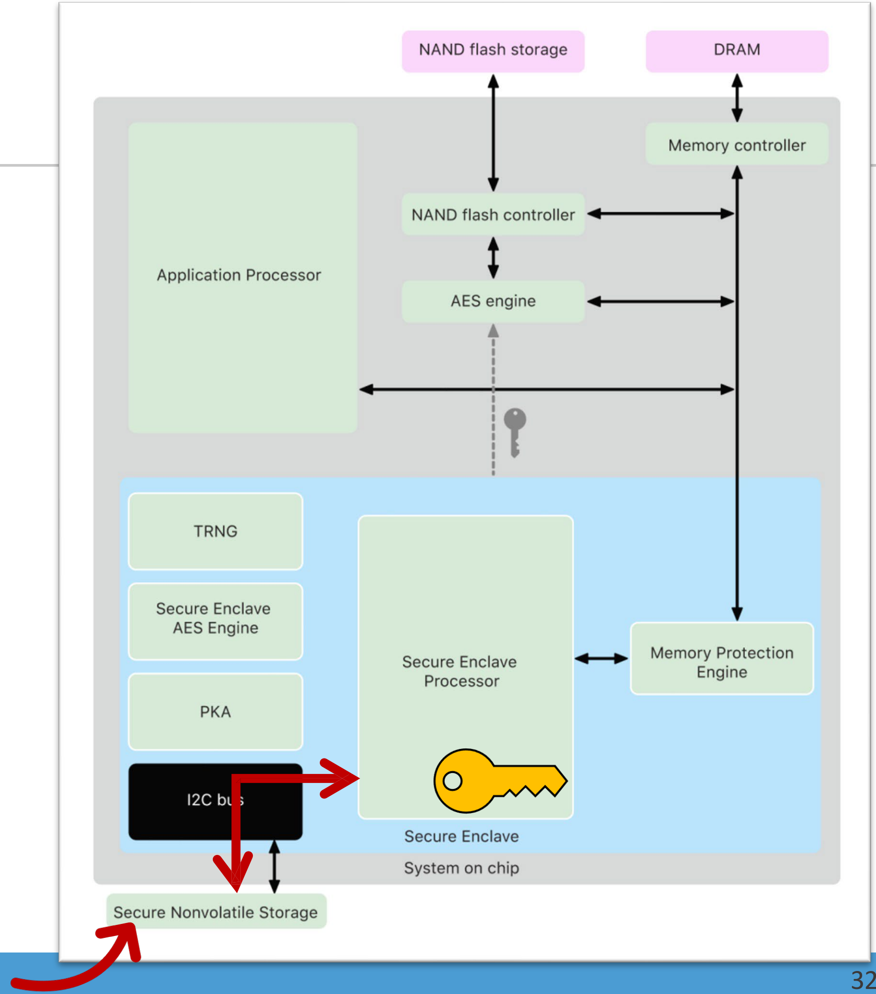
Secure Non-volatile Storage

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

`Passcode + UID -> passcode entropy`

Brute-force has to be performed on the device under attack

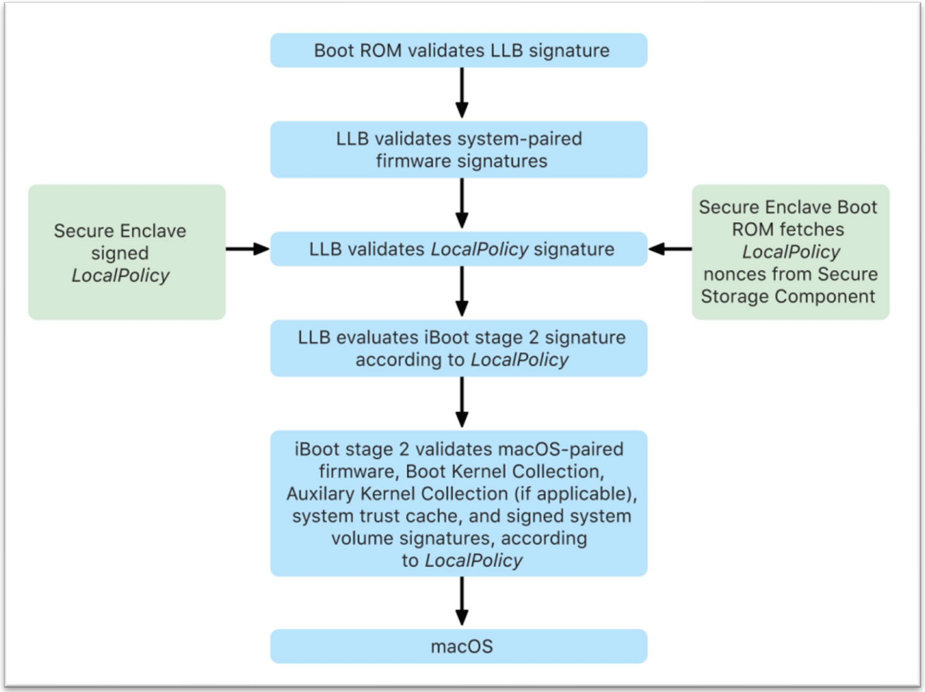
- Escalating time delays
- Erase data when exceeding attempt count



Secure Boot

Similar to TPM but with more constraints

- Each step is signed by Apple to prevent loading non-Apple systems
  - Using Apple Root Certificate authority public key
- 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?

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

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL