Root of Trust

Mengjia Yan

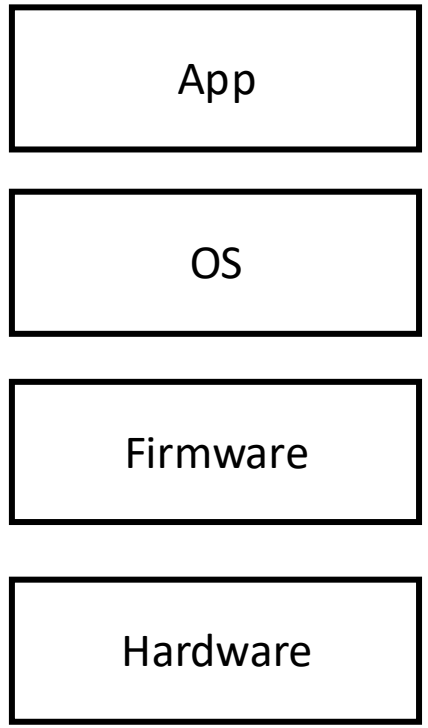
Spring 2026

What Are We Trusting?

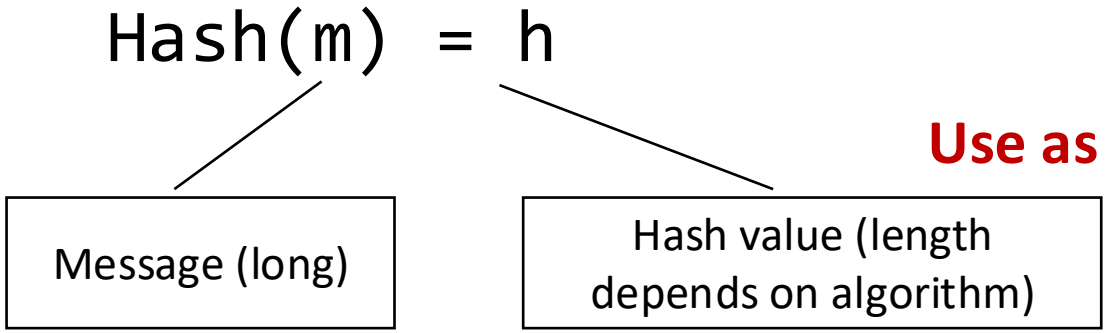
- Example scenario: open a bank app on your phone.

- Application
  - *Is the banking app really the one provided by the bank? Or has it been modified by an attacker?*
- Operating System
  - *Am I really running the OS I think I am? Could malware modify the OS and hide itself from detection?*
- Hardware / Firmware
  - *Is the system actually running on hardware provided by the vendor? Could firmware be modified?*

- **Every layer relies on trusting a deeper layer**



Check Integrity

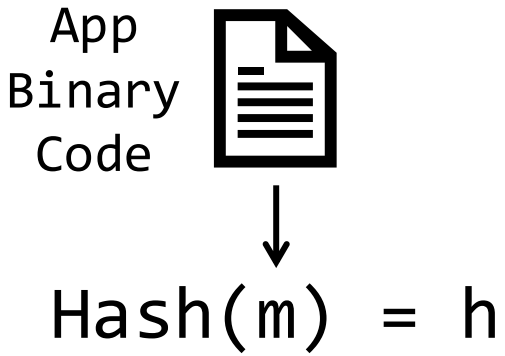


Use as fingerprints

- 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

Application Integrity Verification

- Code signing

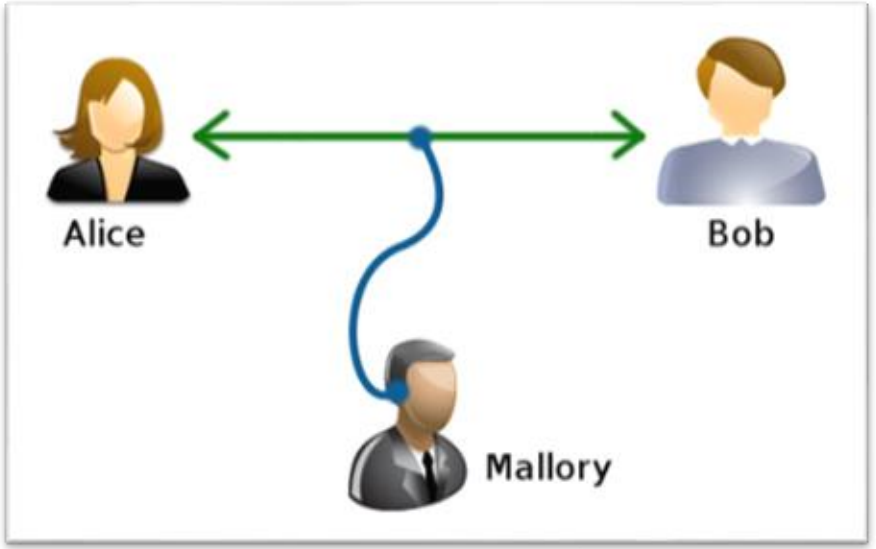


- App store verification
  - Then you need to trust the “app store”, which by itself is an app

- Questions:
  1. Who provides the trusted hash?
  2. Who performs this integrity verification?

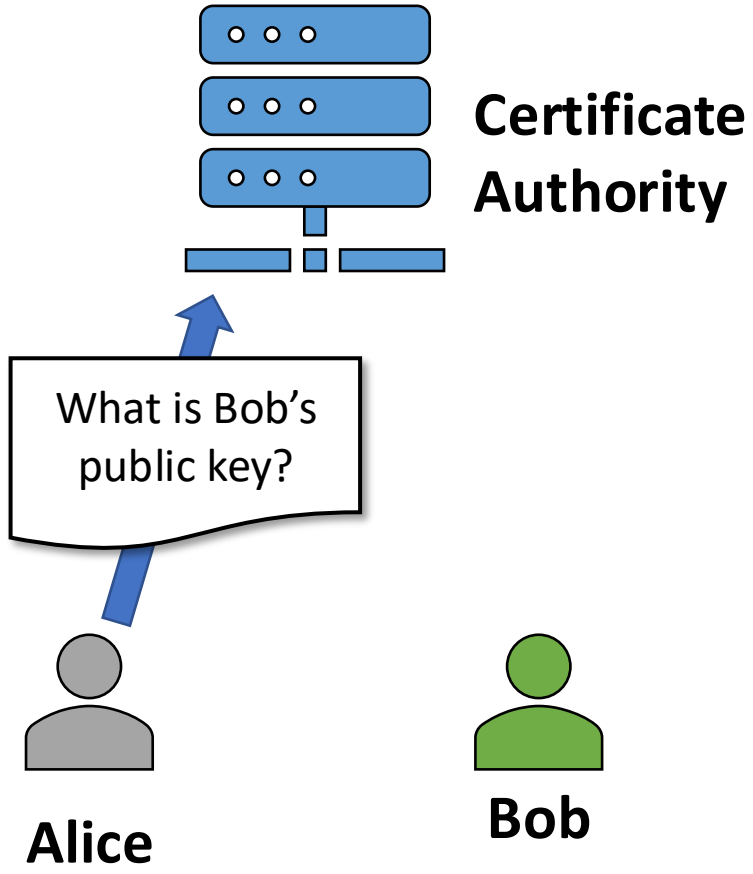
Asymmetric Cryptography (e.g., RSA)

- Goal: authenticate the sender and ensure integrity of the message
- A pair of keys:
  - Private key ($K_{private}$ – kept as secret)
  - Public key (K_{public} – safe to release publicly)
- Computation:
  - `Sign(Hash(message),` $K_{private}$`) = signature`
  - `Verify(message, signature,` K_{public}`) = T/F`
- How to announce and obtain the public key?



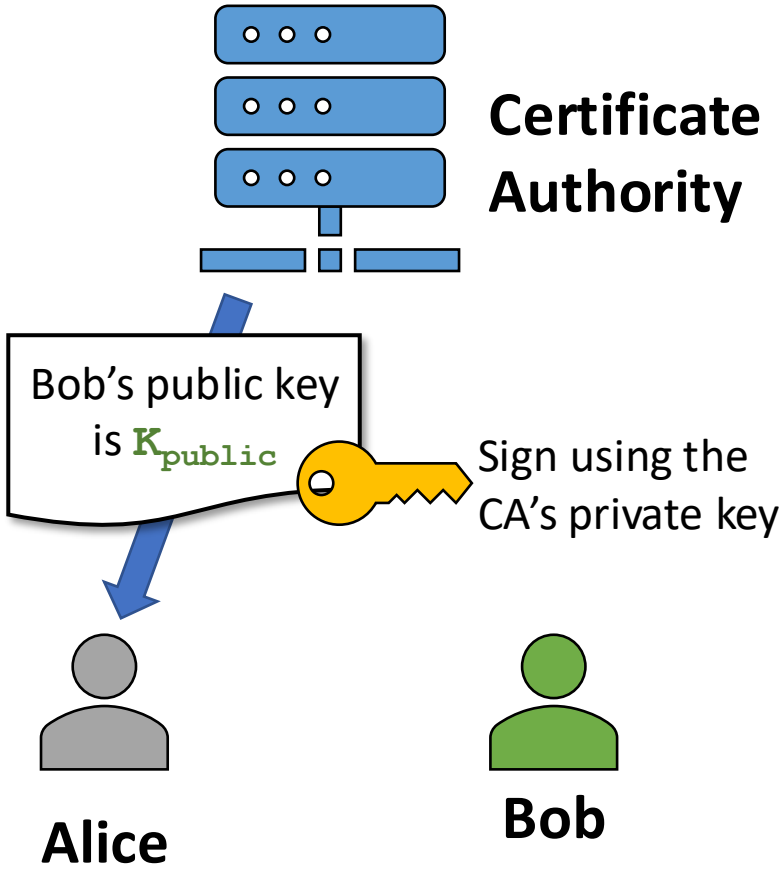
Public Key Infrastructures (PKIs)

- Problem: announce and obtain the public key
- Analogy: public key is like a **government-issued ID**, need to be validated by an authority.
- Bob has a private key $K_{private}$ and wants to claim he corresponds to a public key K_{public}

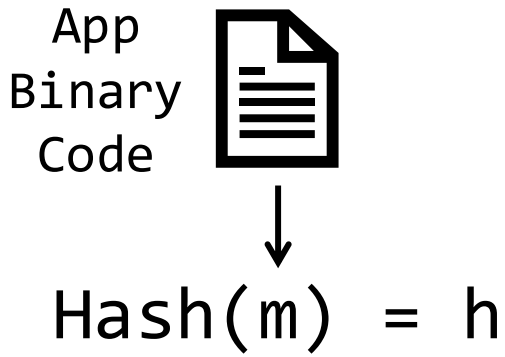


Public Key Infrastructures (PKIs)

- CA signs (Bob, K_public)
- Establish a chain of trust
- **Real-world use cases**: identify website, identify hardware chips/processors



Application Integrity Verification



- Questions:
  1. Who provides the trusted hash?
  2. Who performs this integrity verification?

Digital signature and PKI

The Trust Bootstrapping Problem

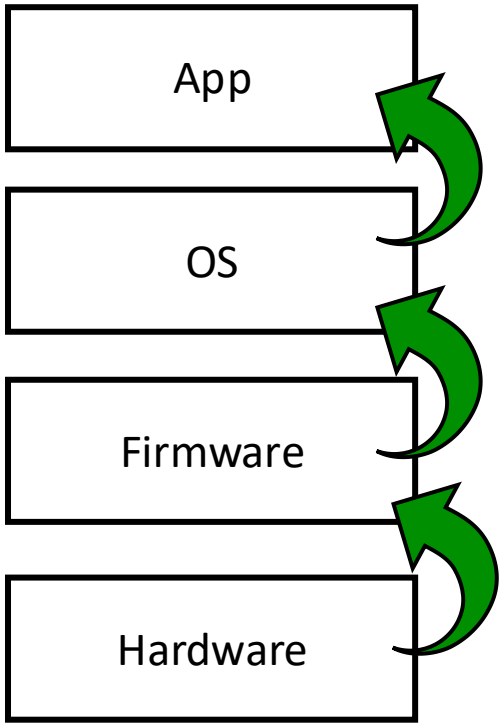
To verify the App

→ need to trust the OS that verifies it

→ need to trust the firmware that verifies OS

→ need to trust the hardware that verifies the firmware

- **Verification requires a trusted starting point: Root of Trust (RoT)**



Root of Trust (RoT)

- ❑ A Root of Trust is a component that performs one or more security-critical functions such as measurement, storage, reporting, or verification, and is inherently trusted. *(TCG specification)*
- ❑Roots of trust are highly reliable hardware, firmware, or software components that perform critical security functions and must be secure by design because other mechanisms depend on them. *(NIST definition)*

- Typical properties:
  - Immutable or highly protected
  - Minimal functionality
  - Used to verify other components

Question: Why is software alone insufficient to establish a Root of Trust?

TCG Roots of Trust Specification: https://trustedcomputinggroup.org/wp-content/uploads/TCG_Roots_of_Trust_Specification_v0p20_PUBLIC_REVIEW.pdf
NIST Roots of Trust https://csrc.nist.gov/projects/Hardware-Roots-of-Trust

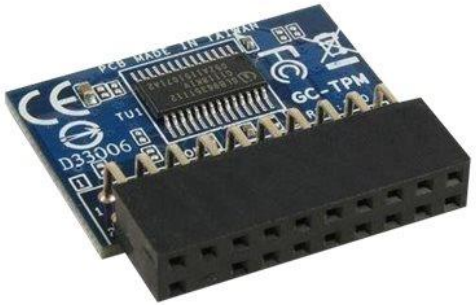
Hardware Root of Trust

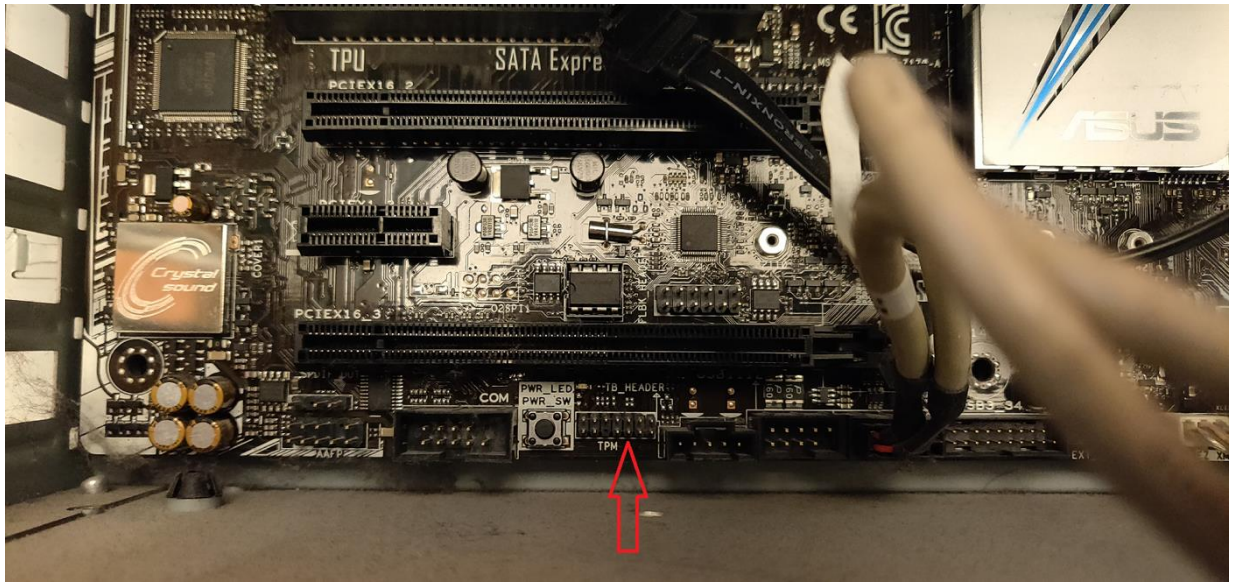
- Hardware RoT performs:
  - Measurement (record system state and compute the hash)
  - Secret storage (tamper resistance to protect keys and secrets)
  - Verification (check whether the computed hash == expected hash)
  - Reporting

- Examples:
  - TPM
  - OpenTitan
  - Apple Enclave

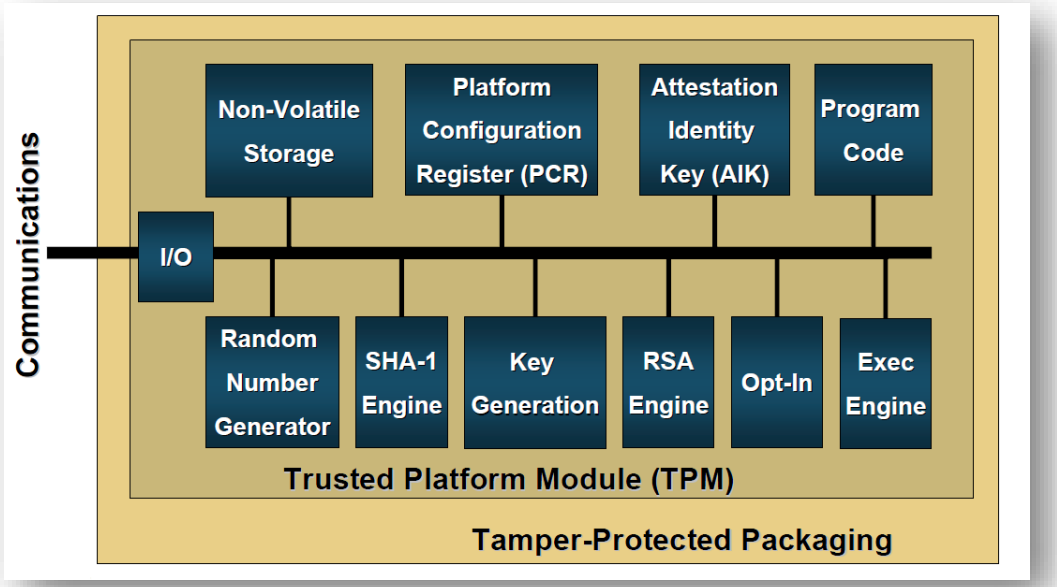
Trusted Platform Module (TPM)

- "*Commoditized* IBM 4758"

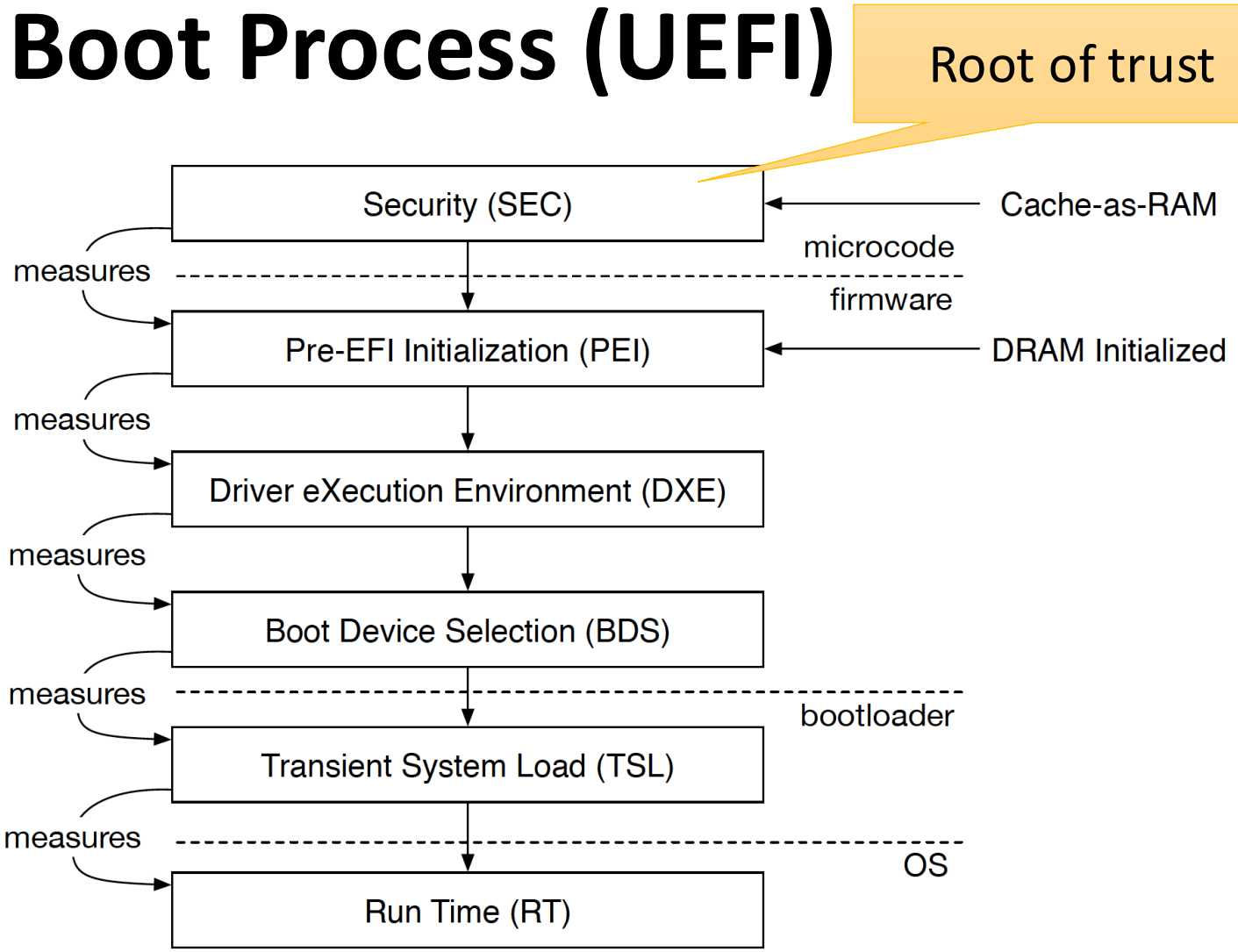


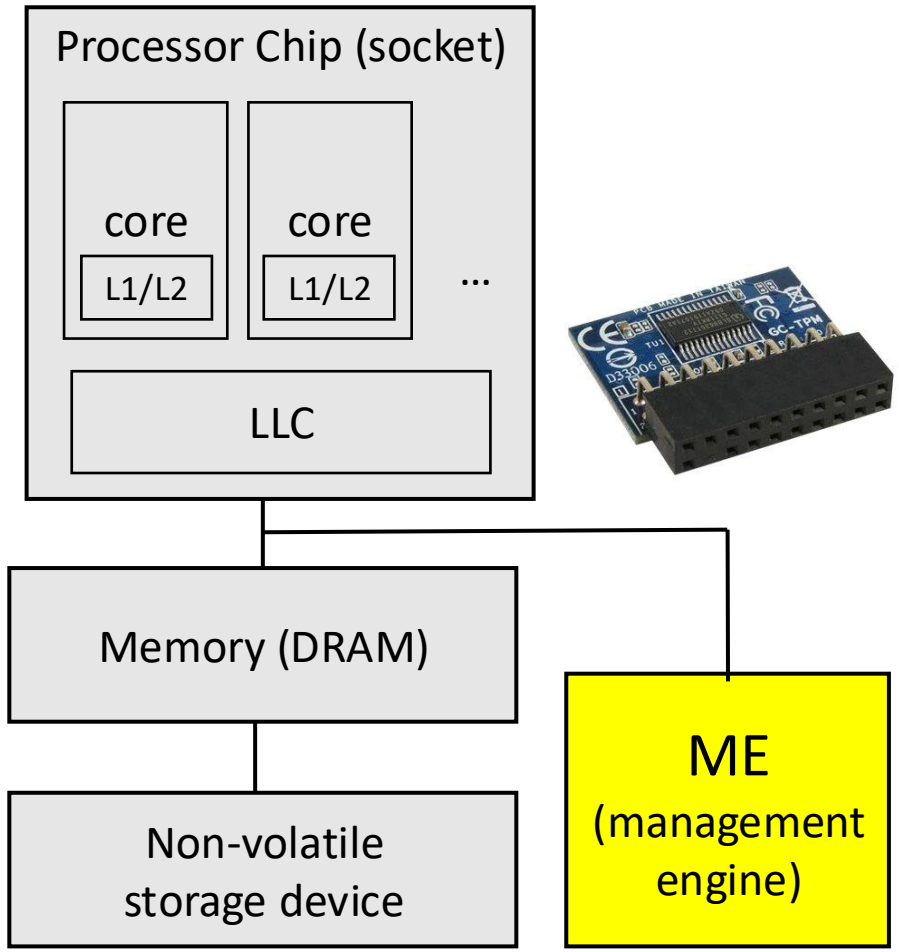


https://scotthelme.co.uk/upgrading-my-pc-with-a-tpm/



Boot Process (UEFI)

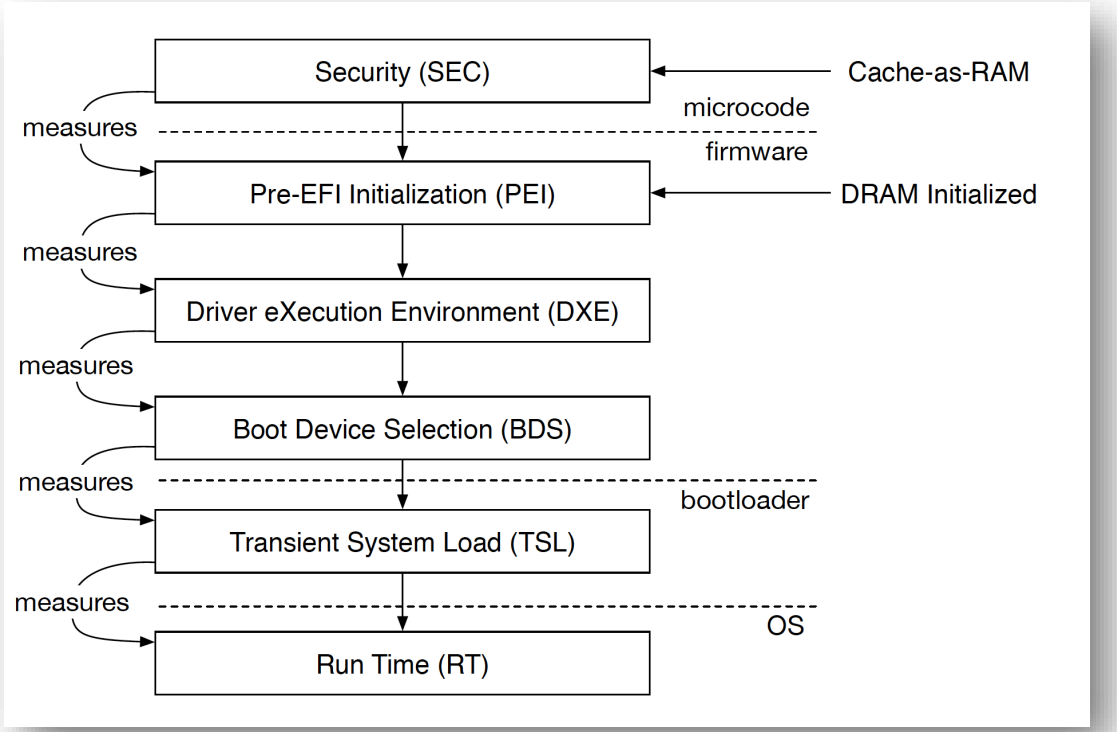




Always measure before executing …

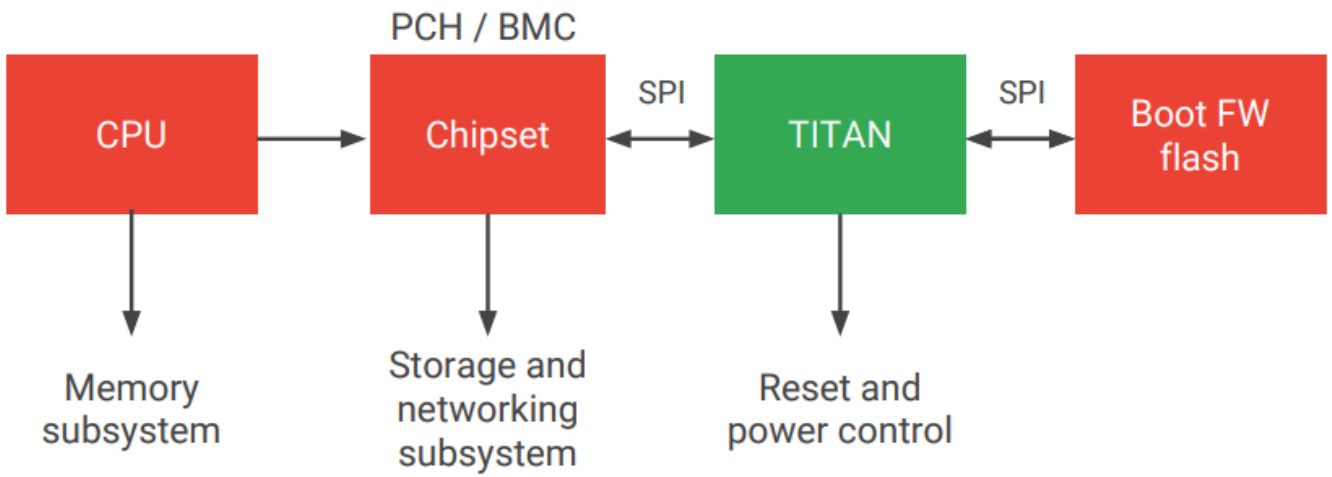
Security Problems of Using TPM

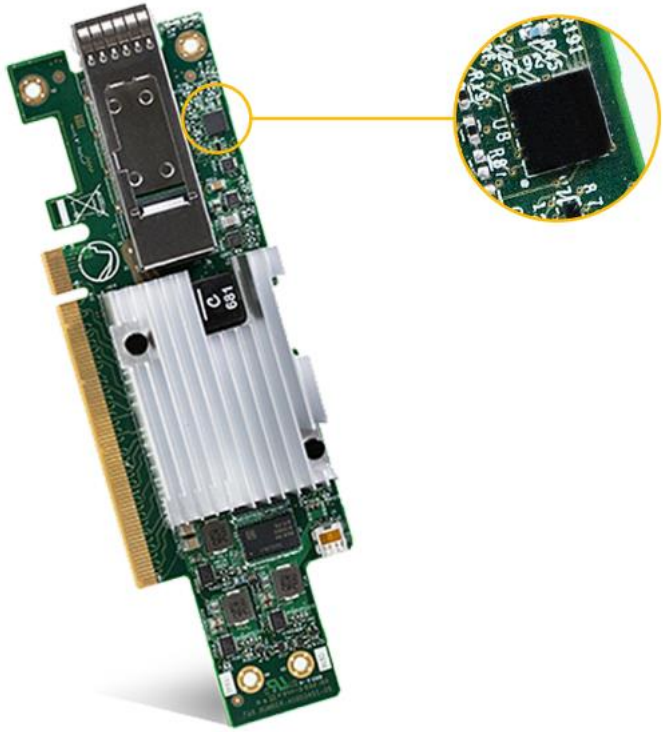
- Assume the first-stage bootloader is securely embedded in motherboard
- Not easy to use with frequent software/kernel update
- Time to check, time to use
- TPM Reset attacks
  - exploiting software vulnerabilities and using software to report false hash values



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

OpenTitan

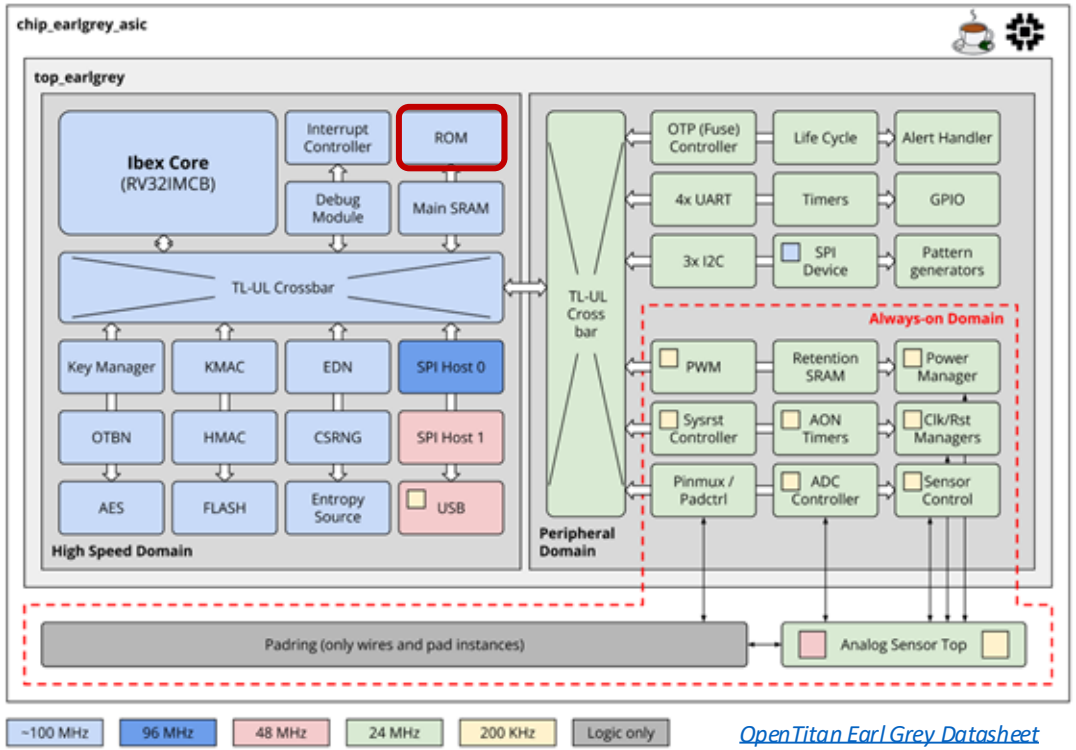




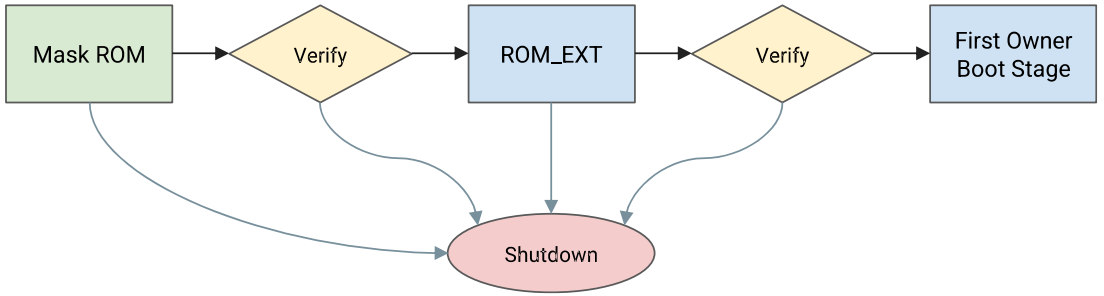
from https://www.hotchips.org/hc30/1conf/1.14_Google_Titan_GoogleFinalTitanHotChips2018.pdf

OpenTitan

- Goal: Establish Root of Trust, validate platform integrity (similar to TPM)
- Boot ROM: Configure critical hardware and verify next boot stage

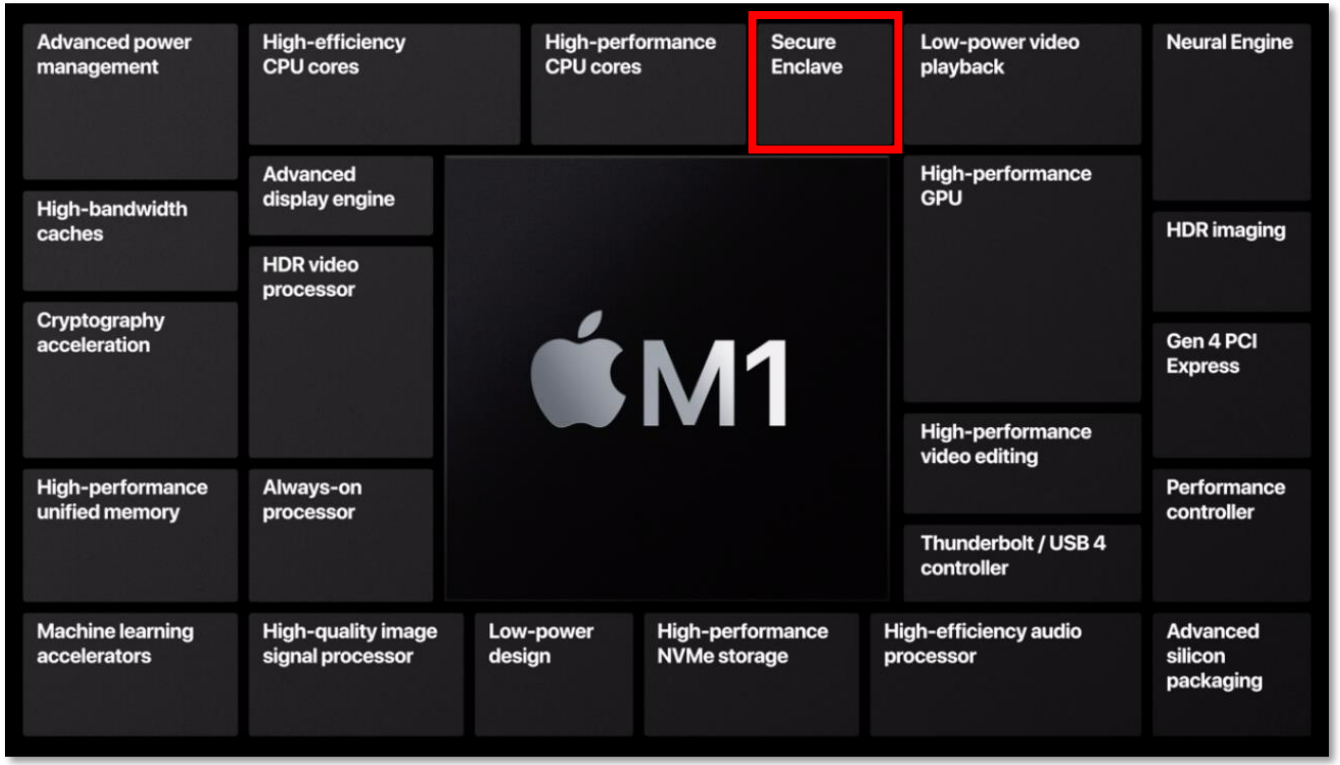


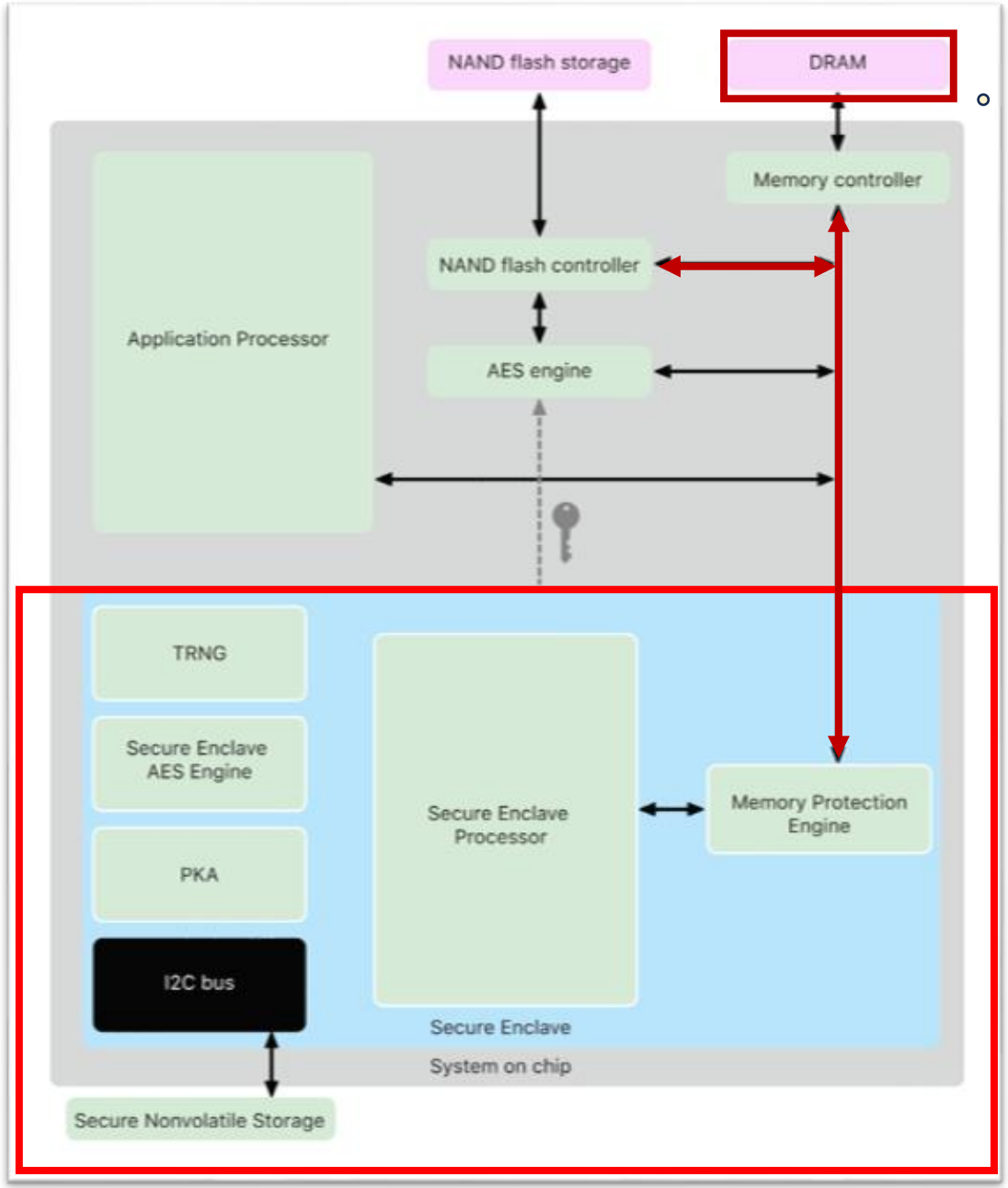
OpenTitan Earl Grey Datasheet



Apple Secure Enclave

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





Shared DRAM? 😟

Encrypt enclave data and only decrypt at the memory protection engine

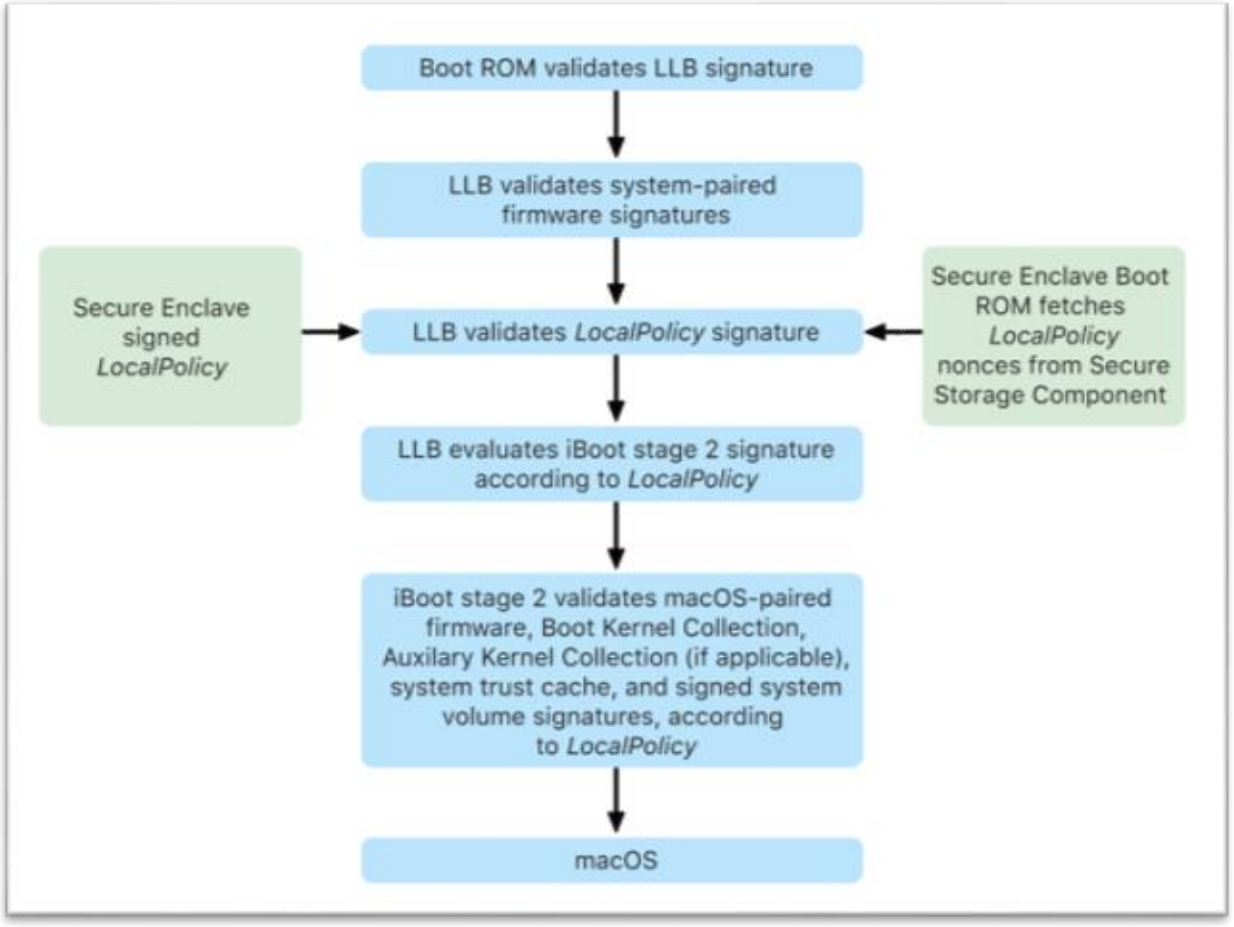
- Only run secure enclave functionality, no user code
- Block vulnerabilities due to software bugs (running L4 microkernel)
- Block uarch side channels

From Apple Platform Security (Page 5-96)

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

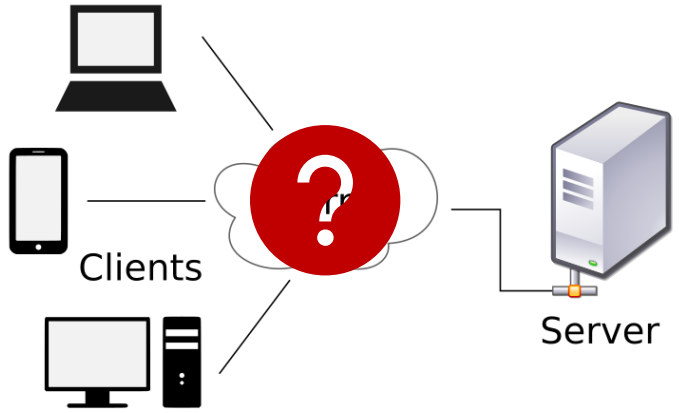


Trust Across the Internet

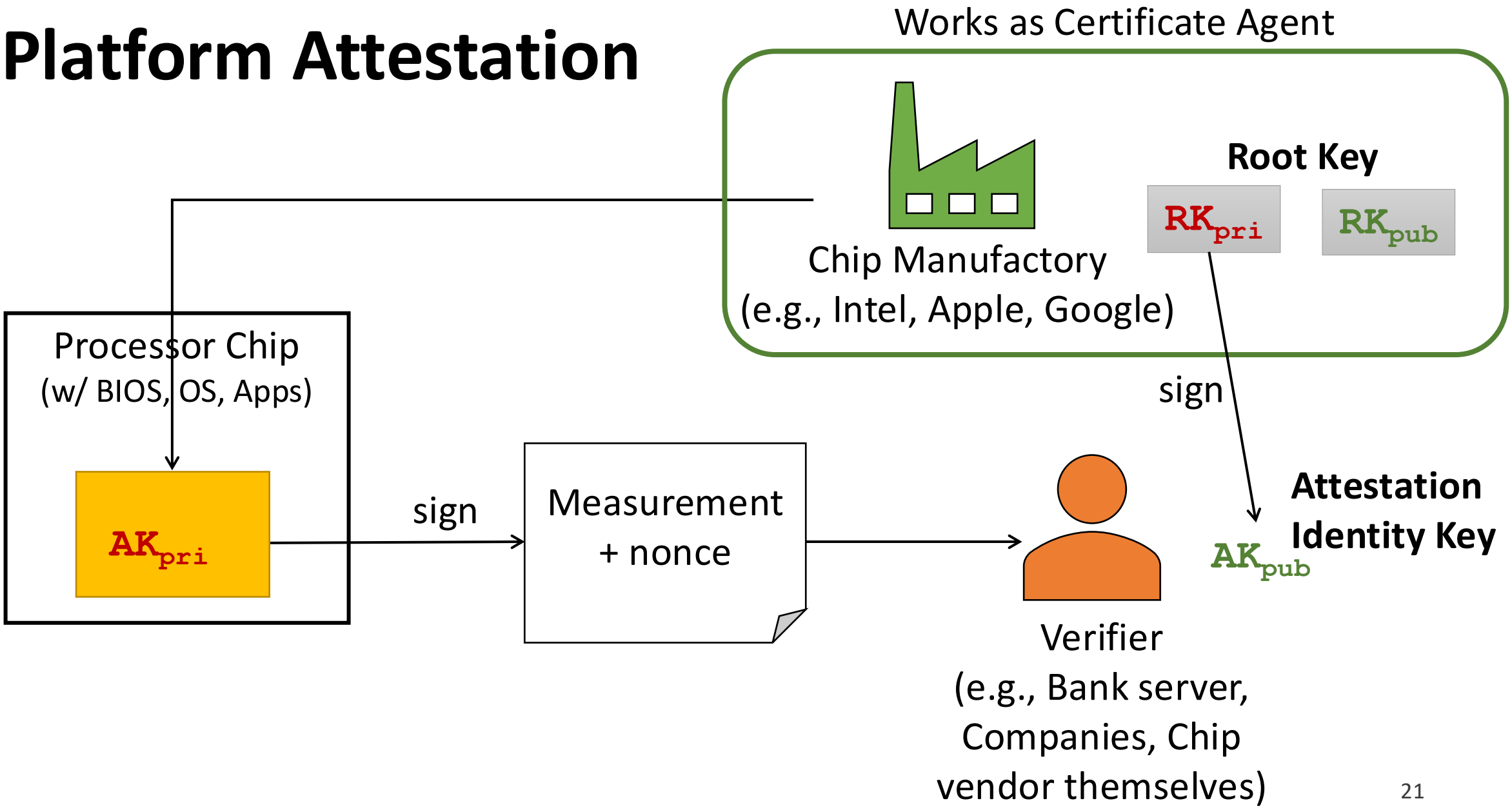
- So far: All these protections happen **inside the device** → Local verifier

- Example scenario: A banking server receives a request from your phone.
- Question: how can the server know that this device is trusted?

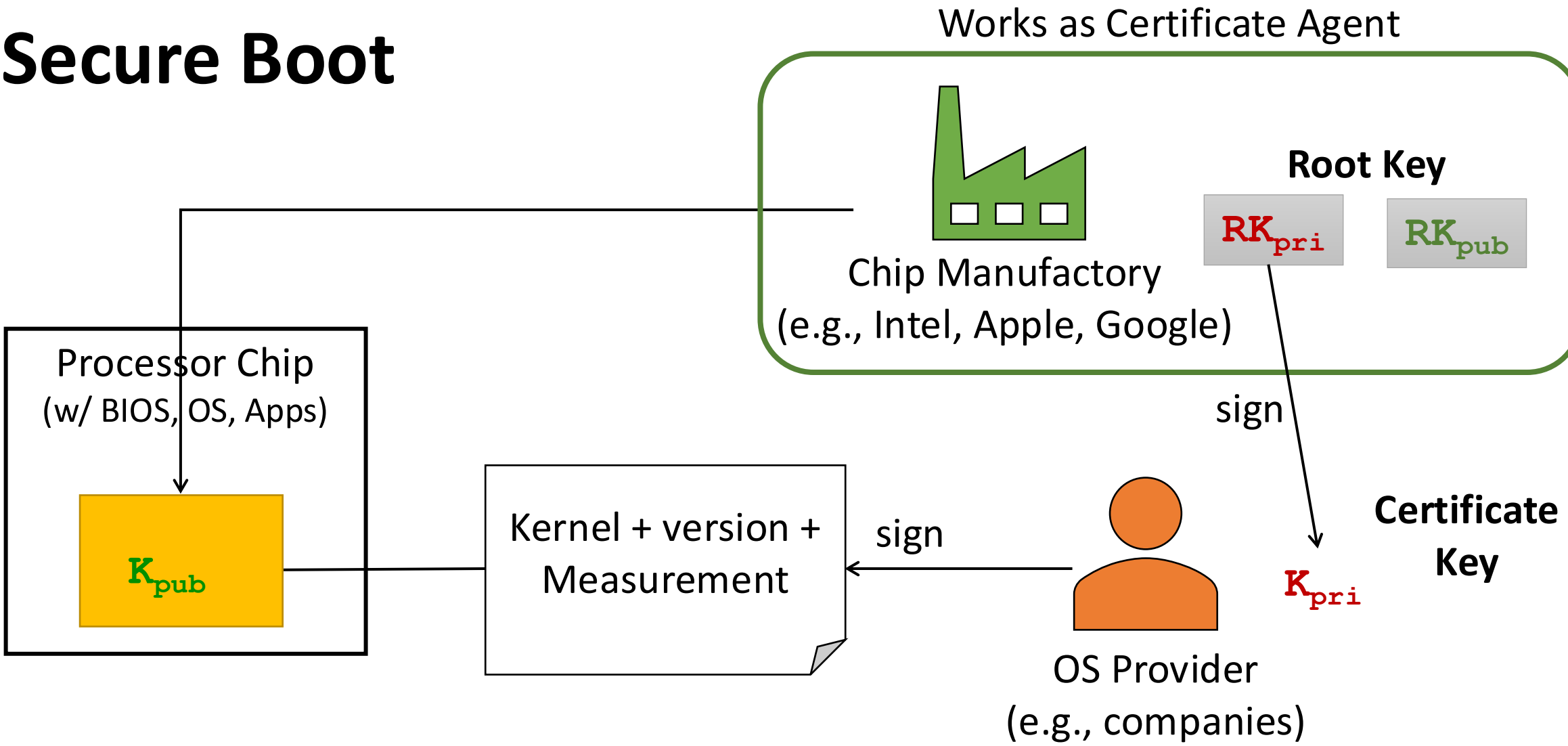
Hardware RoT measures and protects system state.
Remote attestation allows other parties to verify the state.



Platform Attestation



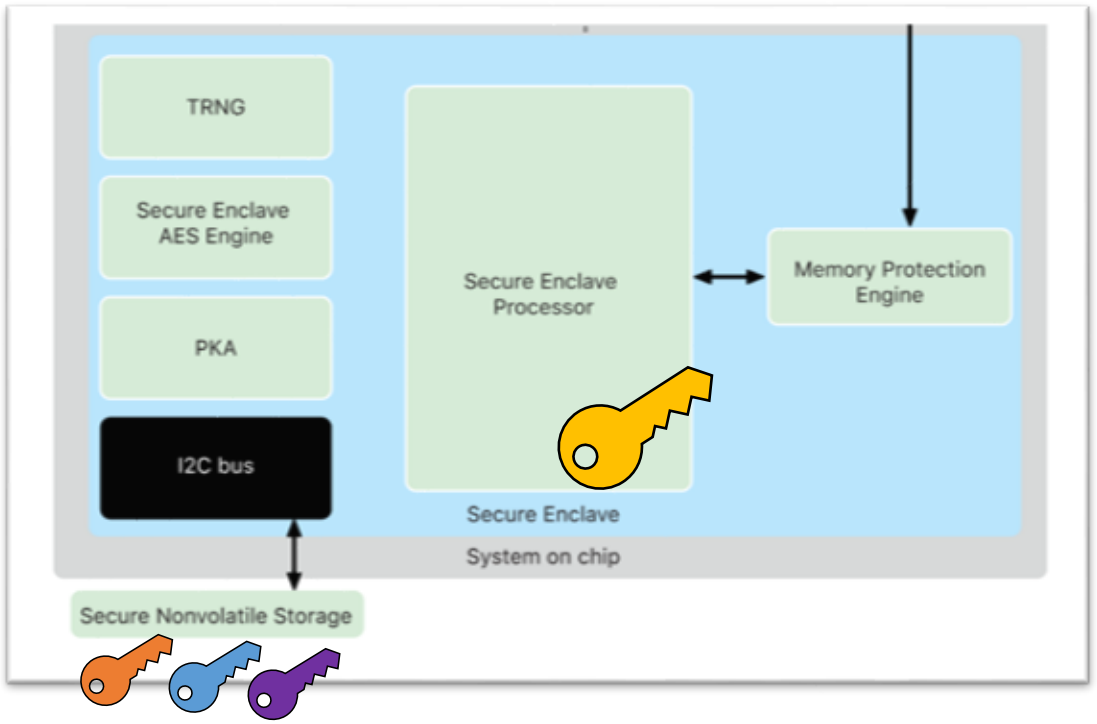
Secure Boot


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

Encrypt using Short Passcode

- How many attempts do we need to brute-force 6-digit passcode?
- How to mitigate brute-force?
- How to deal with attacks who can copy the data across devices and brute-force in parallel?

Bind Crypto Keys to Device



User data encryption **keys**

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

`Passcode + UID -> passcode entropy`

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

Combine with other mitigations:

- Escalating time delays
- Erase data when exceeding attempt count

Next:
Hardware Support for Software Security