Secure Processors In Industry

Mengjia Yan

Spring 2022

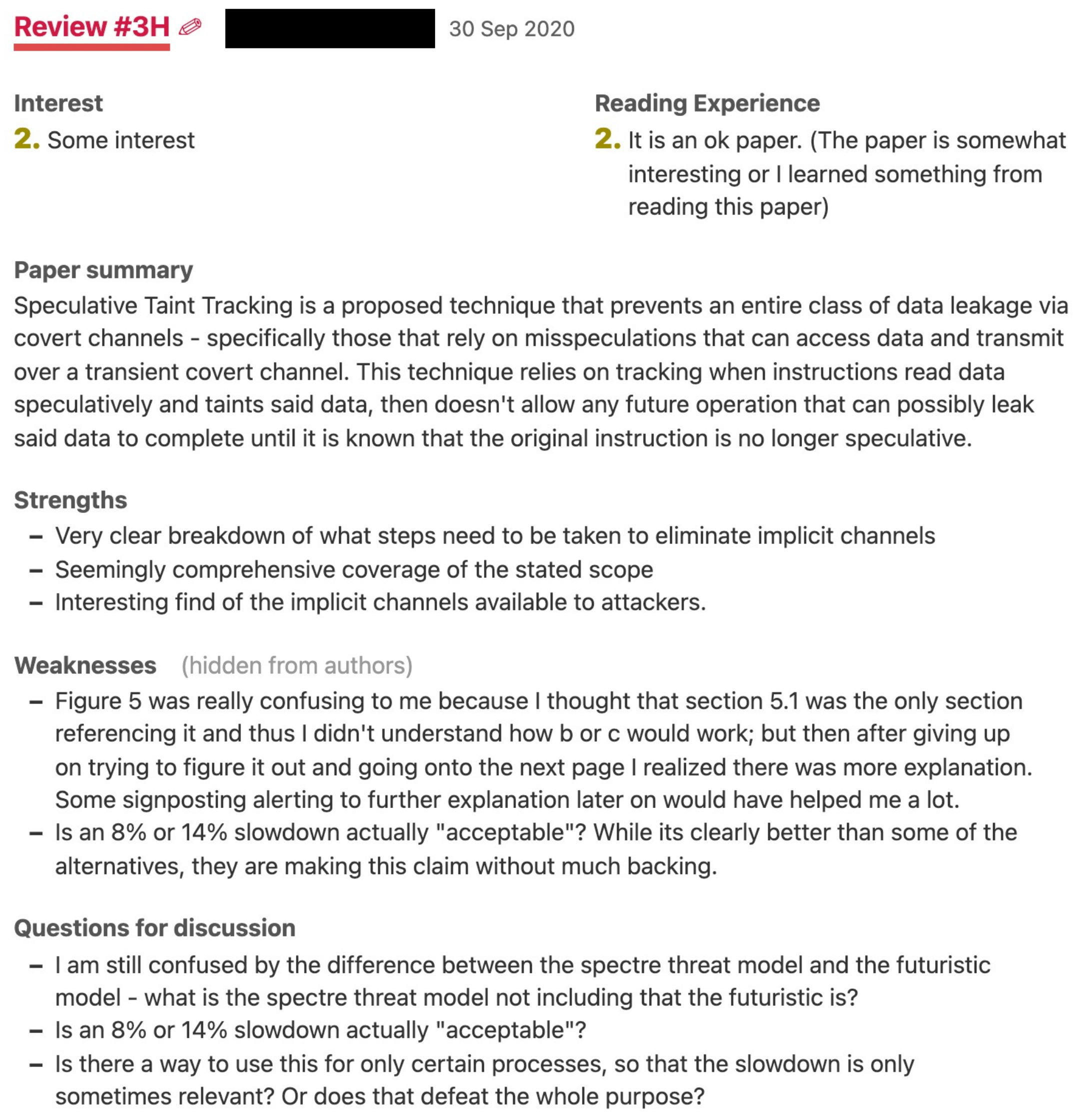
Based on slides from Christopher W. Fletcher and Jakub Szefer

Reminder

- Sign up Piazza

- Recitation
  - Room: 36-153
  - Time: 1-2 pm Friday
  - This Friday: Tutorial on C, Assembly

- HotCRP and paper presentations
  - Demo next Monday: paper bidding
  - Reviews: Signup if you are interested in contributing reviews and reading others' reviews
  - Presentation: Send a private Piazza post if you want to do presentation in pairs



Review #3H [redacted] 30 Sep 2020

Interest
2. Some interest

Reading Experience
2. It is an ok paper. (The paper is somewhat interesting or I learned something from reading this paper)

Paper summary
Speculative Taint Tracking is a proposed technique that prevents an entire class of data leakage via covert channels - specifically those that rely on misspeculations that can access data and transmit over a transient covert channel. This technique relies on tracking when instructions read data speculatively and taints said data, then doesn't allow any future operation that can possibly leak said data to complete until it is known that the original instruction is no longer speculative.

Strengths
- Very clear breakdown of what steps need to be taken to eliminate implicit channels
- Seemingly comprehensive coverage of the stated scope
- Interesting find of the implicit channels available to attackers.

Weaknesses (hidden from authors)
- Figure 5 was really confusing to me because I thought that section 5.1 was the only section referencing it and thus I didn't understand how b or c would work; but then after giving up on trying to figure it out and going onto the next page I realized there was more explanation. Some signposting alerting to further explanation later on would have helped me a lot.
- Is an 8% or 14% slowdown actually "acceptable"? While its clearly better than some of the alternatives, they are making this claim without much backing.

Questions for discussion
- I am still confused by the difference between the spectre threat model and the futuristic model - what is the spectre threat model not including that the futuristic is?
- Is an 8% or 14% slowdown actually "acceptable"?
- Is there a way to use this for only certain processes, so that the slowdown is only sometimes relevant? Or does that defeat the whole purpose?

Outline

- Some Basic Crypto
- Case Studies of Secure Processors

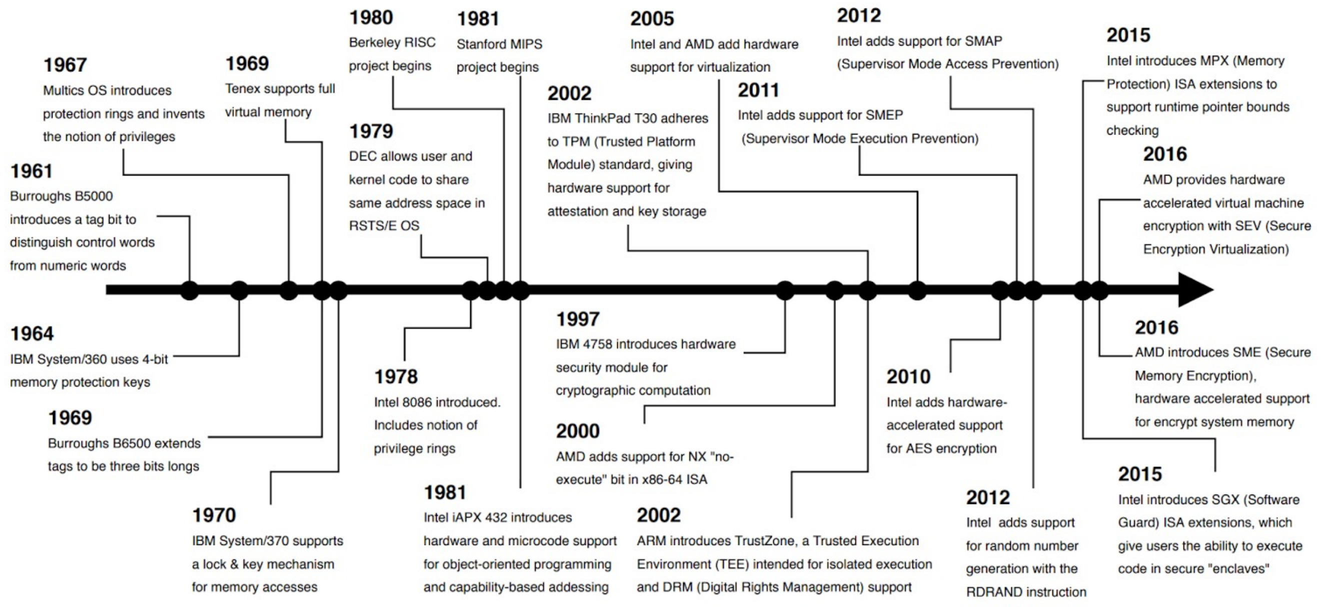

1961
Burroughs B5000 introduces a tag bit to distinguish control words from numeric words
1964
IBM System/360 uses 4-bit memory protection keys
1967
Multics OS introduces protection rings and invents the notion of privileges
1969
Tenex supports full virtual memory
1969
Burroughs B6500 extends tags to be three bits longs
1970
IBM System/370 supports a lock & key mechanism for memory accesses
1978
Intel 8086 introduced. Includes notion of privilege rings
1979
DEC allows user and kernel code to share same address space in RSTS/E OS
1980
Berkeley RISC project begins
1981
Stanford MIPS project begins
1981
Intel iAPX 432 introduces hardware and microcode support for object-oriented programming and capability-based addessing
1997
IBM 4758 introduces hardware security module for cryptographic computation
2000
AMD adds support for NX "no-execute" bit in x86-64 ISA
2002
IBM ThinkPad T30 adheres to TPM (Trusted Platform Module) standard, giving hardware support for attestation and key storage
2002
ARM introduces TrustZone, a Trusted Execution Environment (TEE) intended for isolated execution and DRM (Digital Rights Management) support
2005
Intel and AMD add hardware support for virtualization
2010
Intel adds hardware-accelerated support for AES encryption
2011
Intel adds support for SMEP (Supervisor Mode Execution Prevention)
2012
Intel adds support for SMAP (Supervisor Mode Access Prevention)
2012
Intel adds support for random number generation with the RDRAND instruction
2015
Intel introduces MPX (Memory Protection) ISA extensions to support runtime pointer bounds checking
2015
Intel introduces SGX (Software Guard) ISA extensions, which give users the ability to execute code in secure "enclaves"
2016
AMD provides hardware accelerated virtual machine encryption with SEV (Secure Encryption Virtualization)
2016
AMD introduces SME (Secure Memory Encryption), hardware accelerated support for encrypt system memory

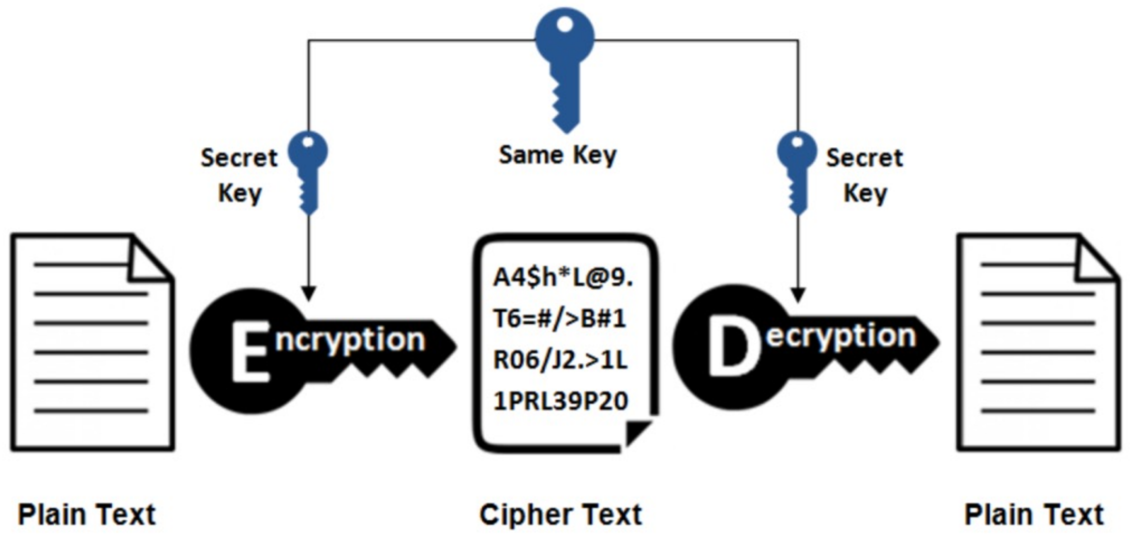
Figure 1: Sixty years of hardware support for security

Security Goals

- Confidentiality
- Integrity
- Identification
- Availability

Example: Lost device

Symmetric Cryptography



- One-time-pad (OTP)

c: cipher text

p: plain text

k: key

```
def enc():
  for i from range(0, L):
    c[i] = p[i] ^ k[i]

def dec():
  for i from range(0, L):
    p[i] = c[i] ^ k[i]
```

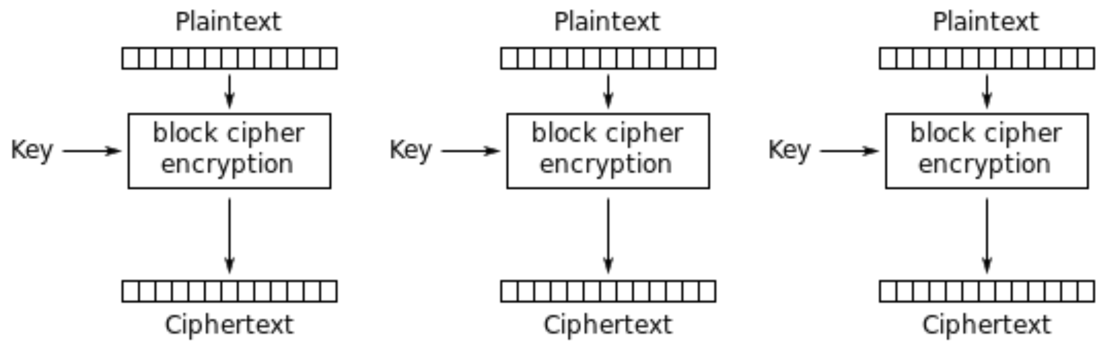
What if we use the key for multiple times?

How about encrypting arbitrary length message? Any problems?

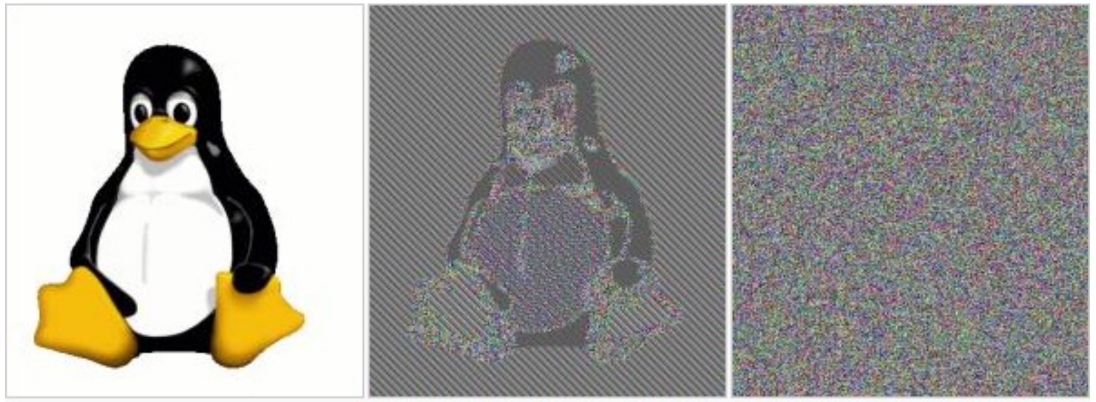
Block ciphers (e.g., DES, AES)

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

ECB IS NOT RECOMMENDED

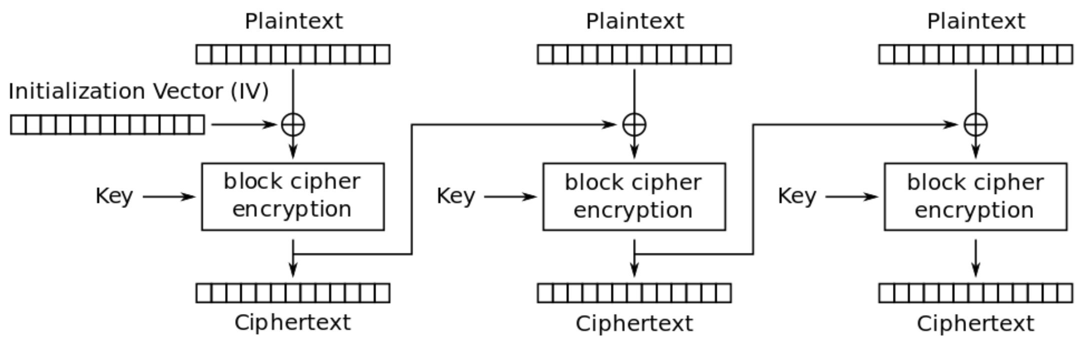


Electronic Codebook (ECB) mode encryption

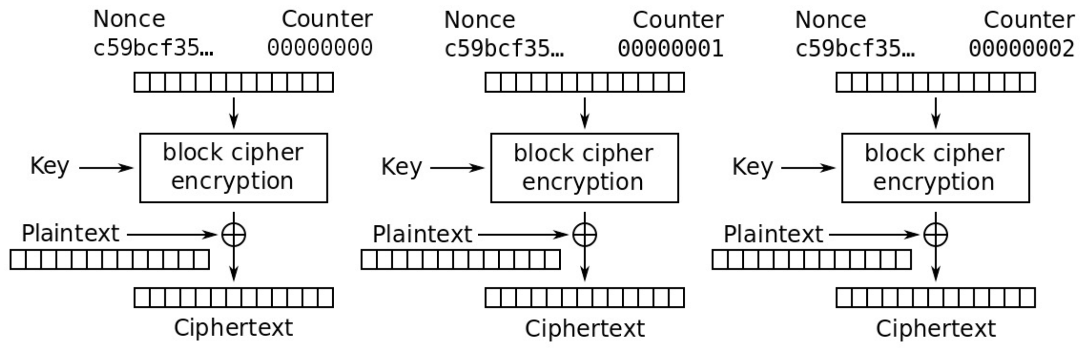


What if the attacker sees P[i] = P[j], where i and j are block id?

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



Cipher Block Chaining (CBC) mode encryption



Counter (CTR) mode encryption

Compare the two schemes:

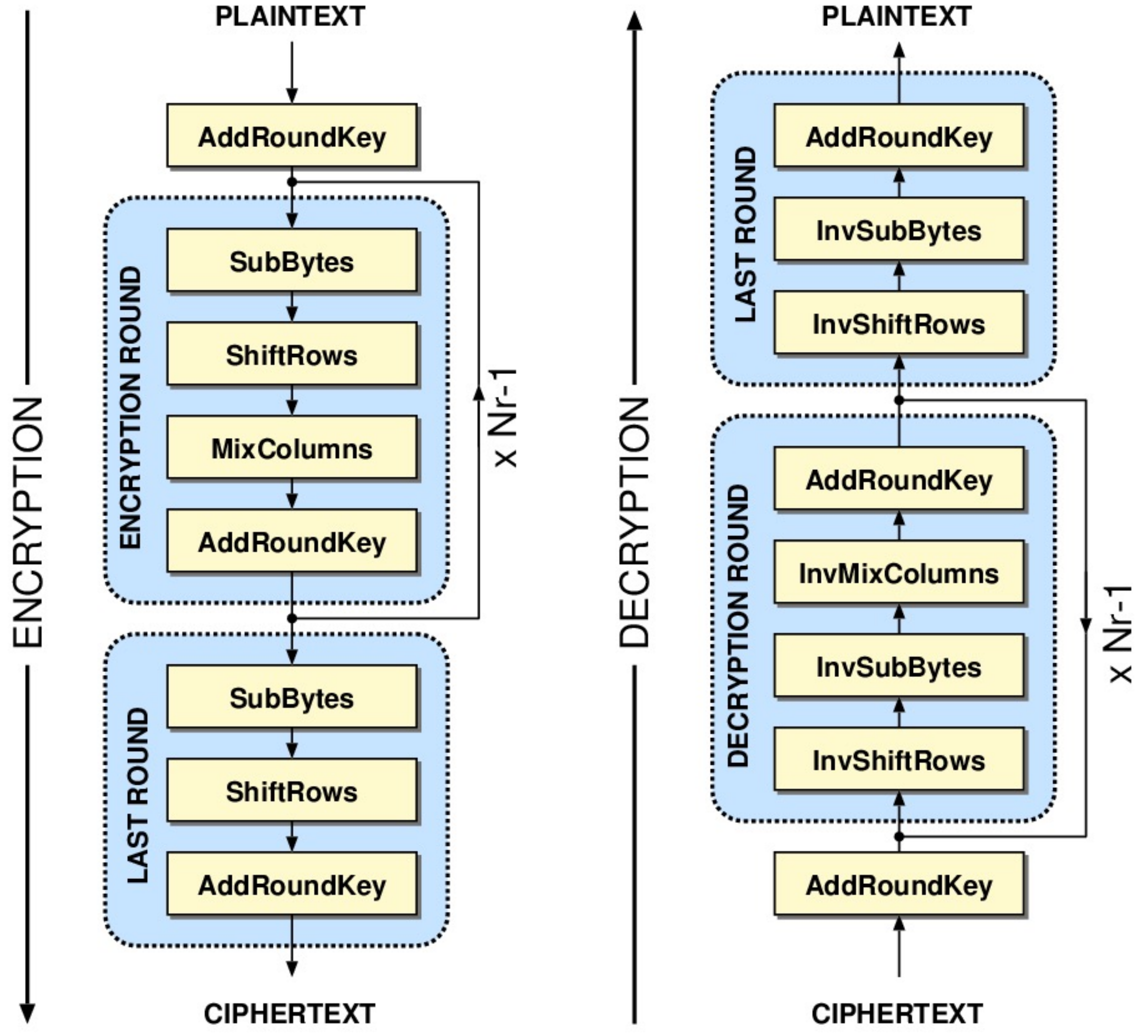
1) What if a block is tampered or a block is lost?
2) Which one has potential higher performance if you have multiple computation units?

When applying AES to encrypt memory, what can be used as Counter?

AES Implementation

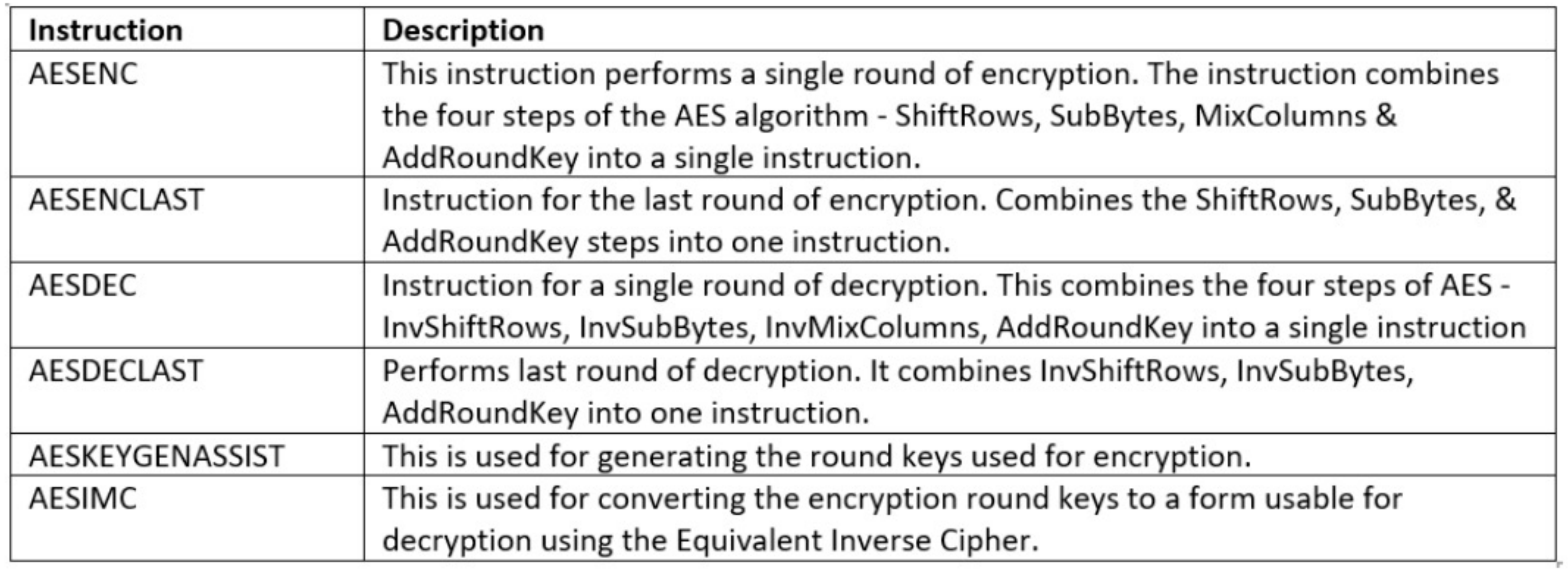
- Goal: "approximate" pseudorandom permutations
- 10, 12, 14 rounds depending on key size
- AddRoundKey: One-time pad using round key
- SubBytes: Sbox lookup

Side channel vulnerability. Will talk in future lectures



AES-NI Instruction Set

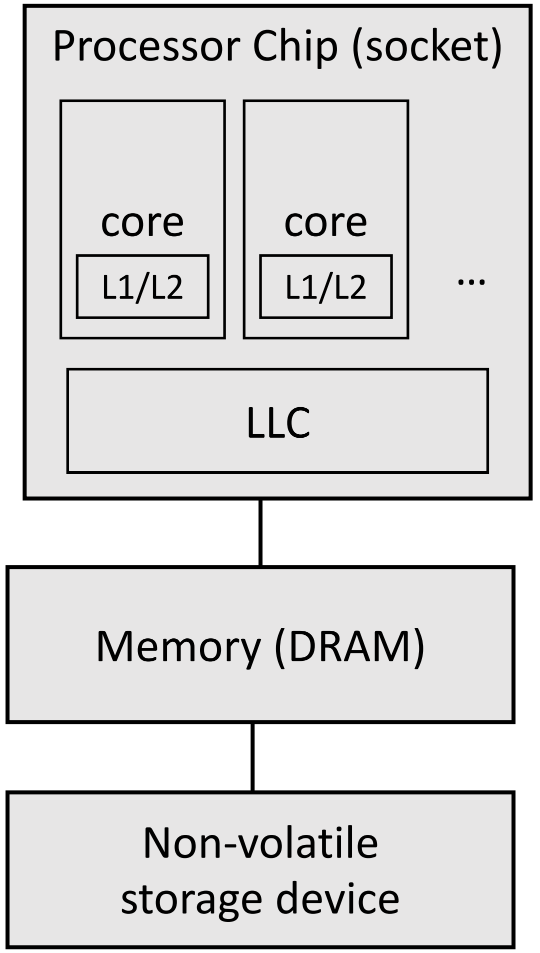
- Short for Advanced Encryption Standard New Instructions
- New instruction + Hardware acceleration
- Both Intel and AMD released supported CPUs around 2010/2011



Instruction	Description
AESENC	This instruction performs a single round of encryption. The instruction combines the four steps of the AES algorithm - ShiftRows, SubBytes, MixColumns & AddRoundKey into a single instruction.
AESENCLAST	Instruction for the last round of encryption. Combines the ShiftRows, SubBytes, & AddRoundKey steps into one instruction.
AESDEC	Instruction for a single round of decryption. This combines the four steps of AES - InvShiftRows, InvSubBytes, InvMixColumns, AddRoundKey into a single instruction
AESDECLAST	Performs last round of decryption. It combines InvShiftRows, InvSubBytes, AddRoundKey into one instruction.
AESKEYGENASSIST	This is used for generating the round keys used for encryption.
AESIMC	This is used for converting the encryption round keys to a form usable for decryption using the Equivalent Inverse Cipher.

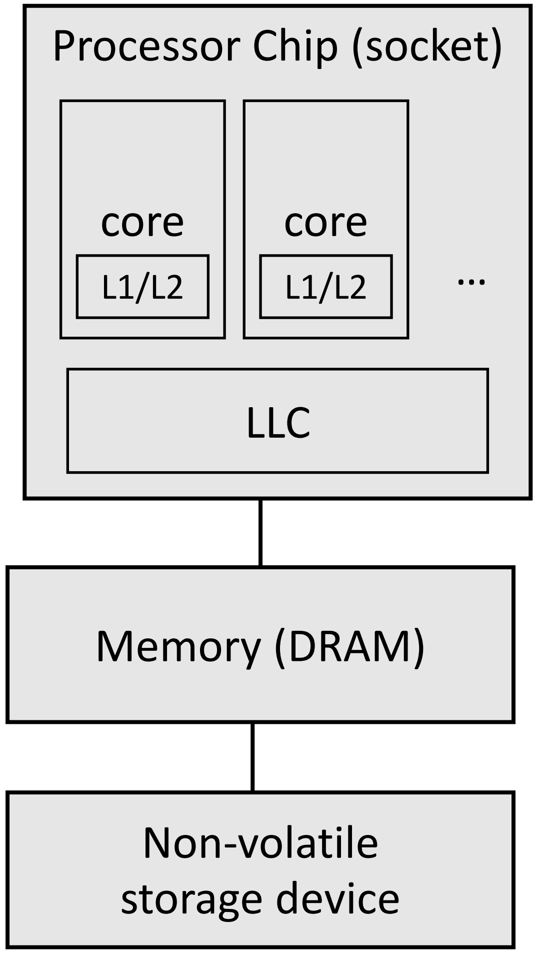
Now back to lost devices

- Is password login sufficient?
- Where encryption should be used?
- Where to store secret keys?

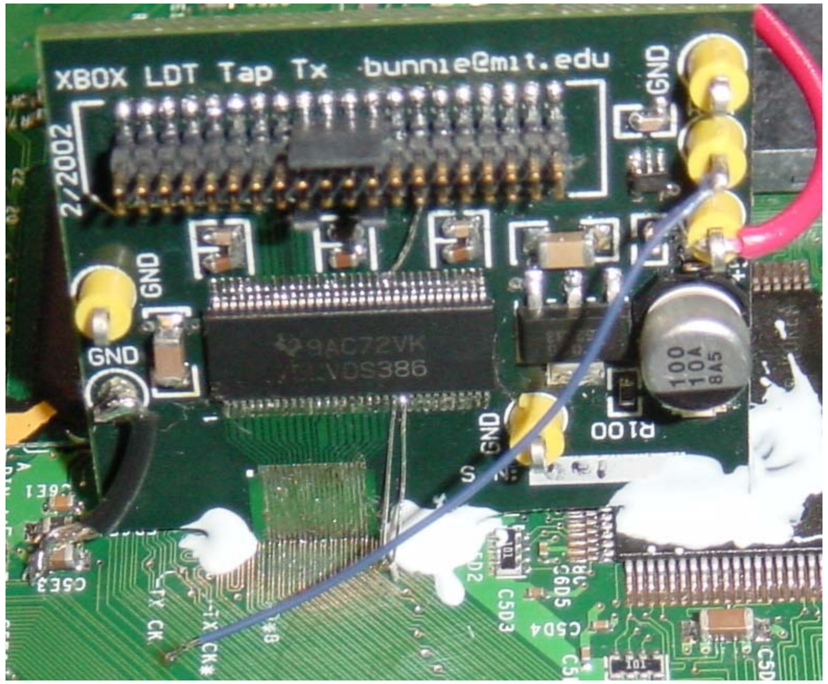


Is Encrypting Disk Sufficient?

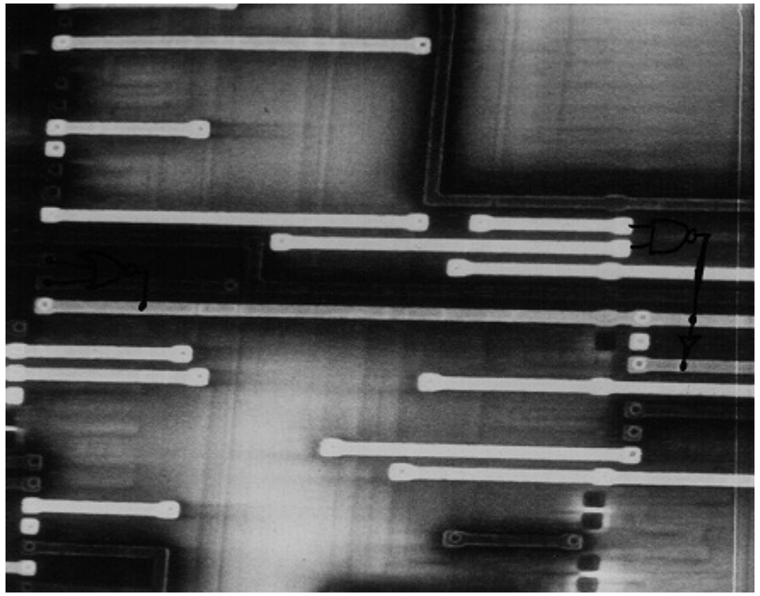
- **Cold boot attacks** to circumvent software-based disk encryption
- An example: *https://www.youtube.com/watch?v=vWHDqBV9yGc*
- How to deal with it?
- Data remanence in SRAM and DRAM



Physical Attack Examples



Tap board used to intercept data transfer over Xbox's HyperTransport bus
from *http://www.xenatera.com/bunnie/proj/anatak/xboxmod.html*

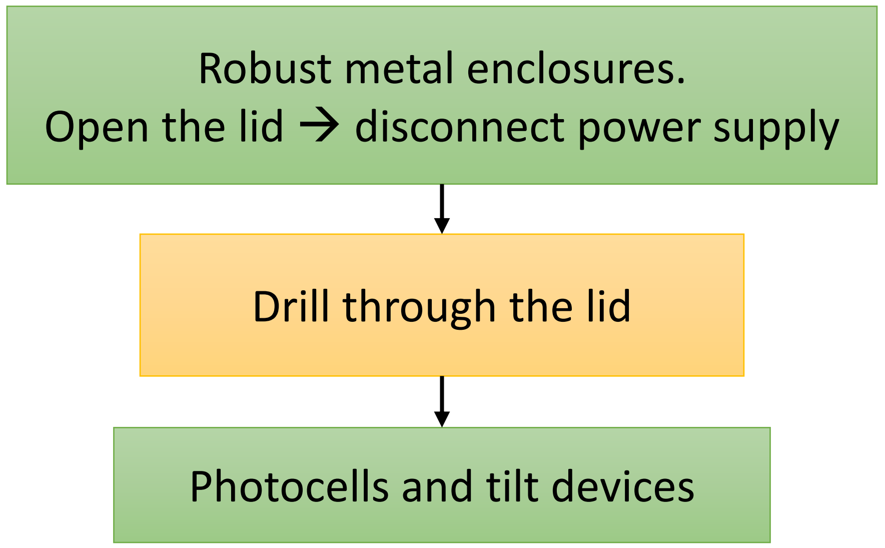


IC analysis. Extract information from a Flash ROM storage cell
from http://testequipmentcanada.com/VoltageContrastPaper.html

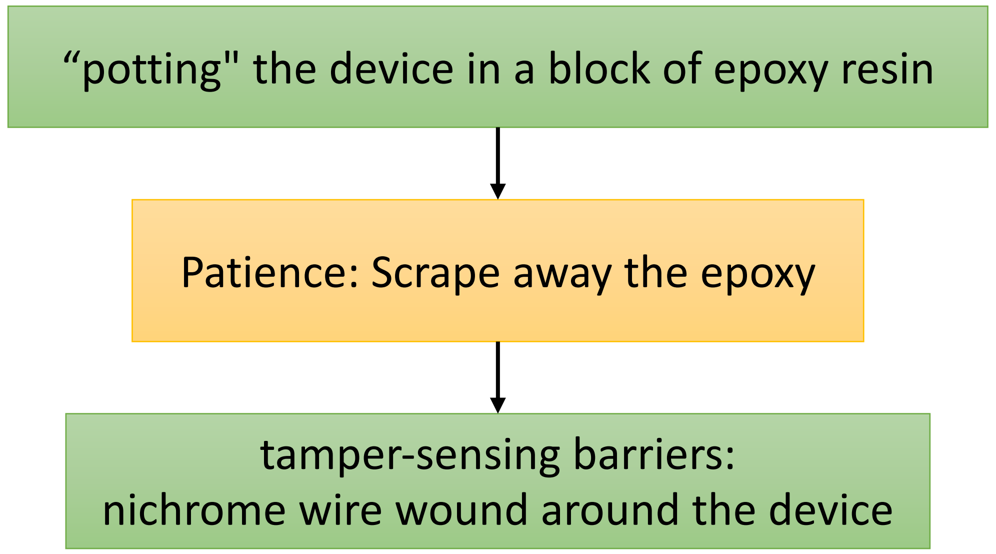
Physical Tamper Resistance

- Standalone security modules to protect cryptographic keys and personal identification numbers (PINs)
- A history lesson of physical security by IBM 4758

Tampering Detection


Robust metal enclosures.
Open the lid → disconnect power supply
Drill through the lid
Photocells and tilt devices

Tampering Evident


"potting" the device in a block of epoxy resin
Patience: Scrape away the epoxy
tamper-sensing barriers:
nichrome wire wound around the device

IBM 4758 Secure Co-Processor

- Memory remanence
  - constant movement of values from place to place
- Cold boot
  - detects changes of temperature
- X-ray
  - a radiation sensor
- Power side channels
  - Solid aluminium shielding and a low-pass filter (a Faraday cage)

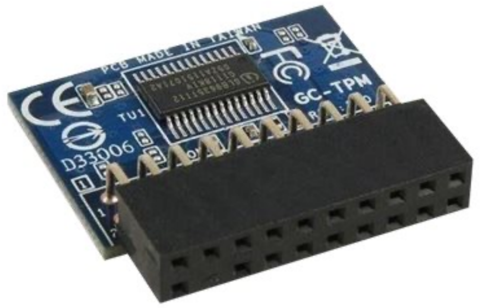
Photo of IBM 4758 Cryptographic Coprocessor (courtesy of Steve Weingart) from *https://www.cl.cam.ac.uk/~rnc1/descrack/ibm4758.html*

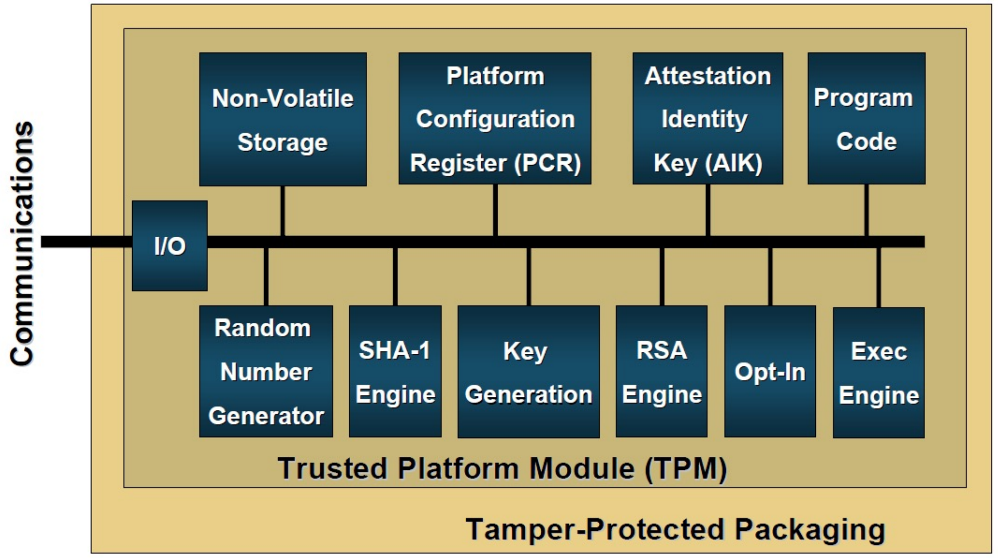
Expensive. Other secure processors only focus on a limited set of physical attacks.

Trusted Platform Module (TPM)

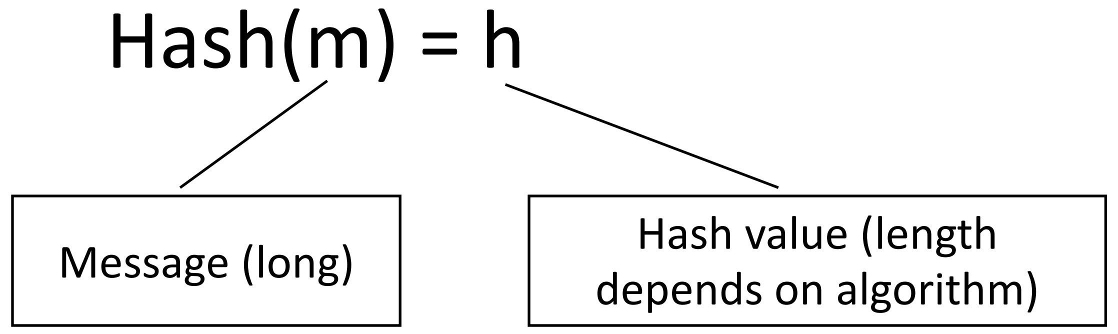
- “Commoditized IBM 4758”
- Standard LPC interface – attaches to commodity motherboards
- Weaker computation capability

- Uses:
  - Disk encryption and password protection
  - Verify platform integrity (firmware+OS)





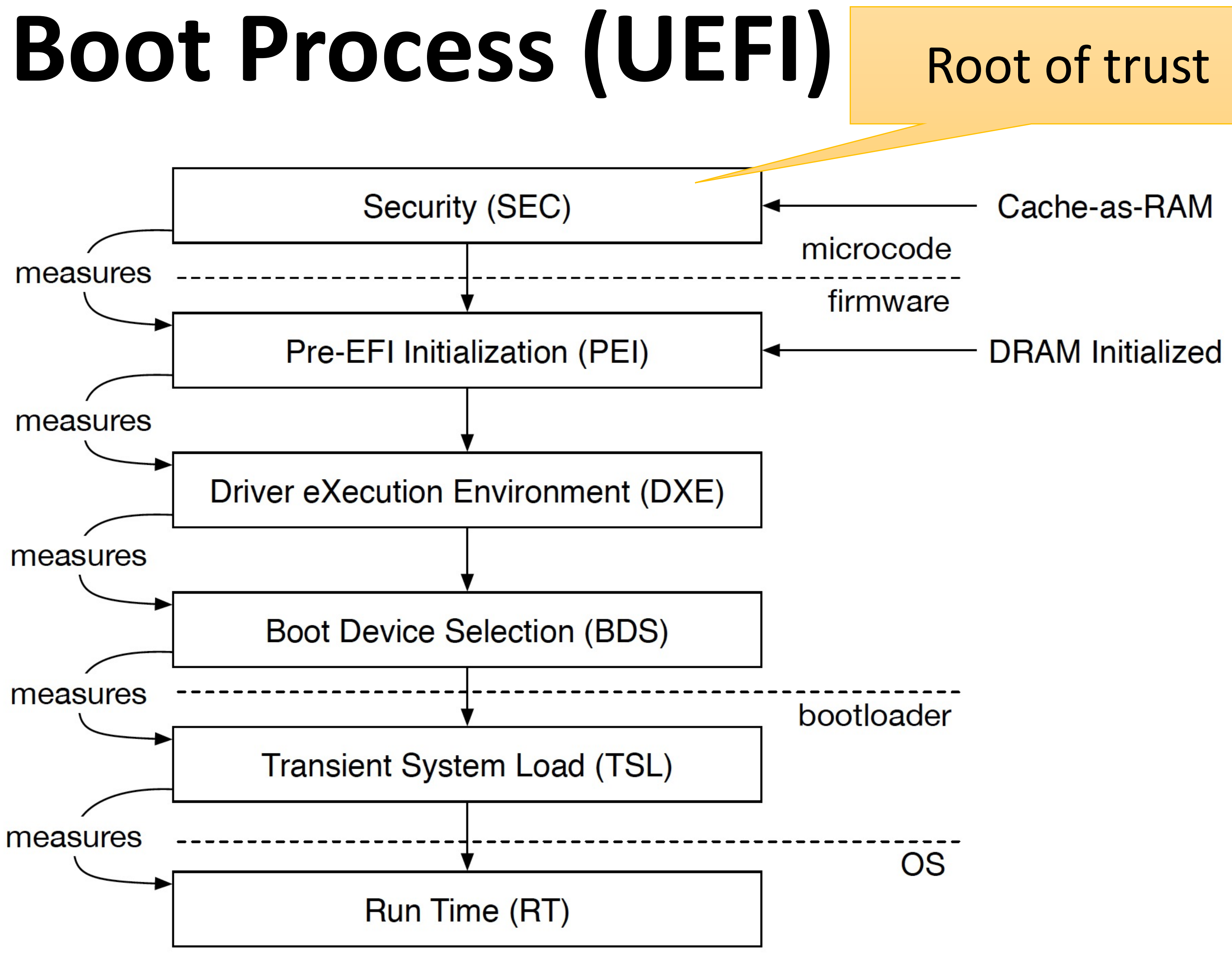
Integrity Verification

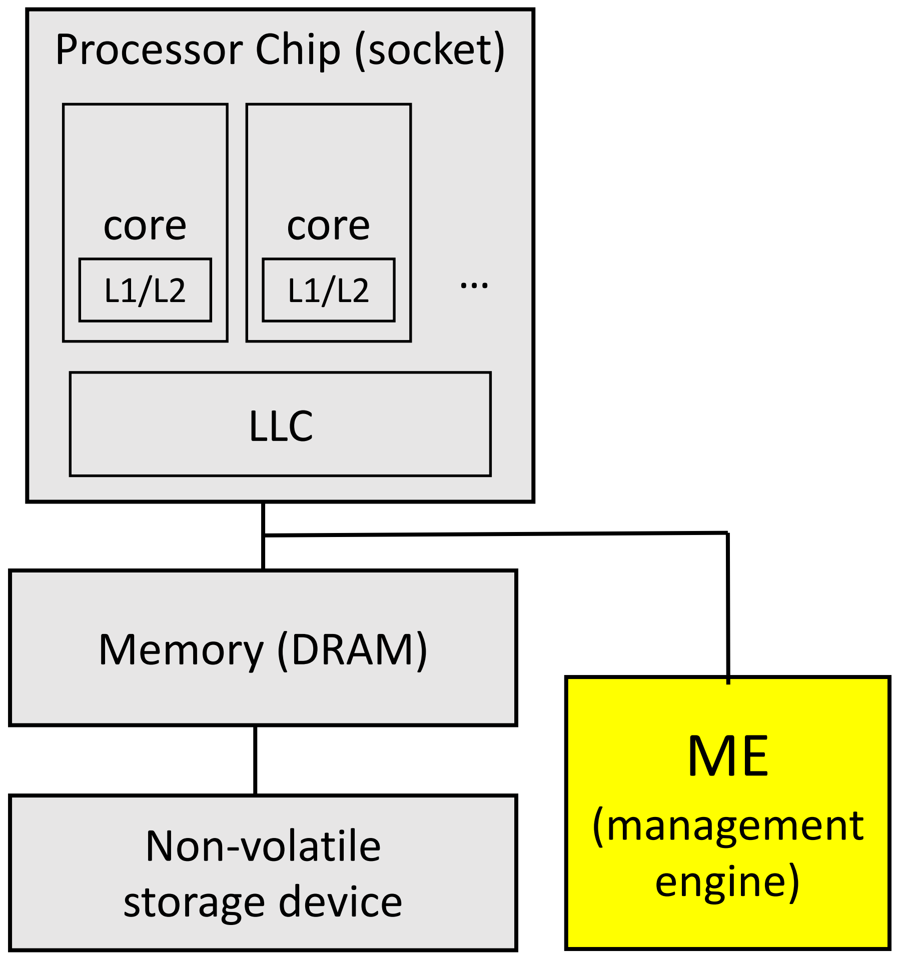


Use as fingerprints

- One-way hash
  - Practically infeasible to invert, 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

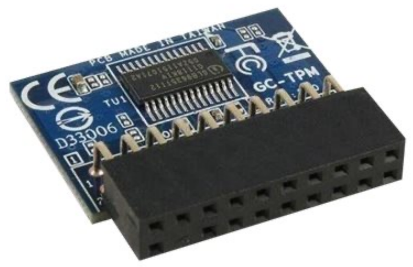
Boot Process (UEFI)



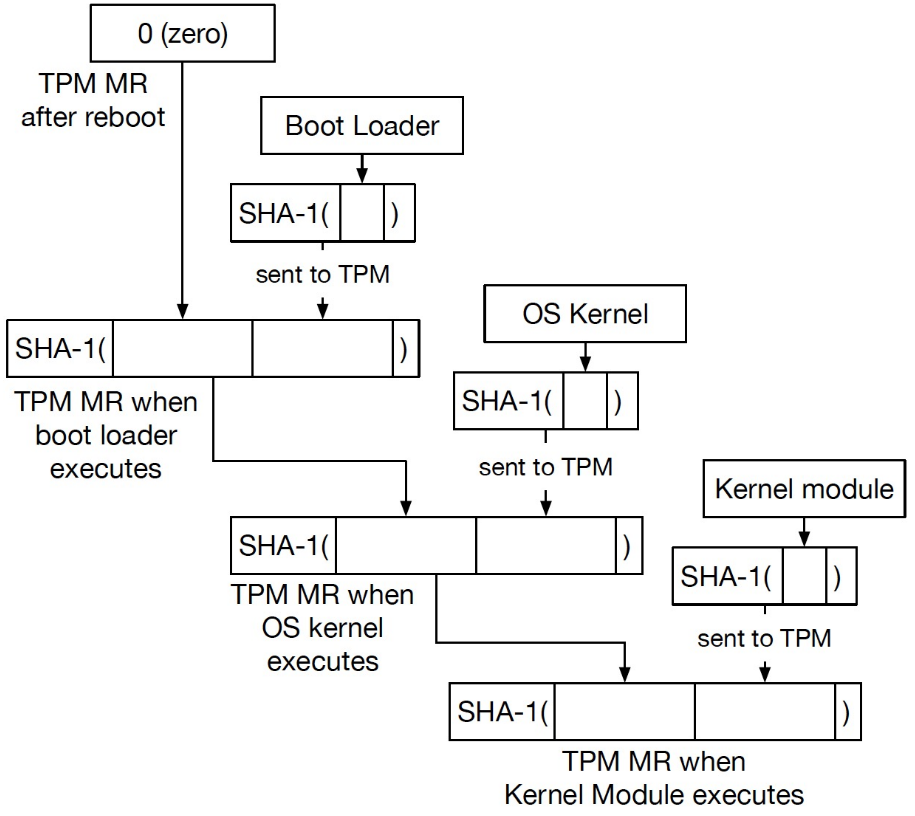


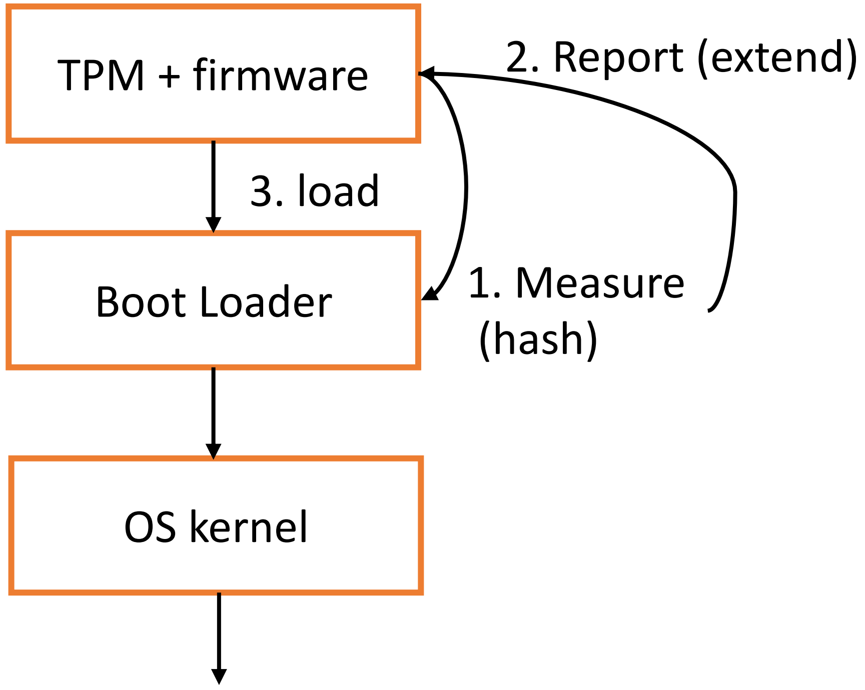
How to perform the measurement?

Secure Boot using TPM



- Static root of trust for measurement (SRTM)



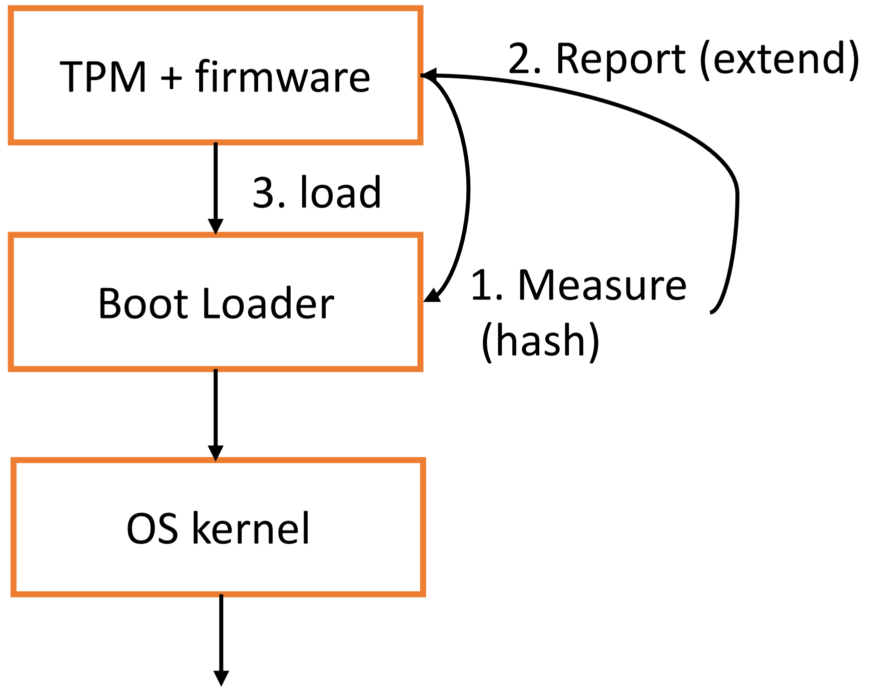


Compared to expected values locally or submitted to a remote attestor.

PCR: platform configuration register

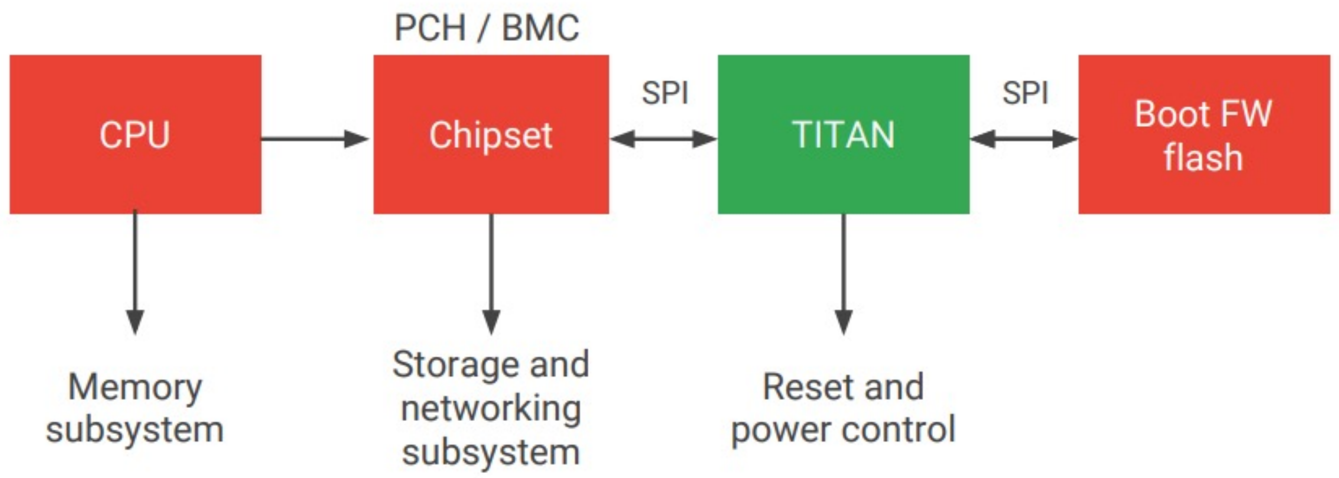
Security Vulnerabilities of Using TPM

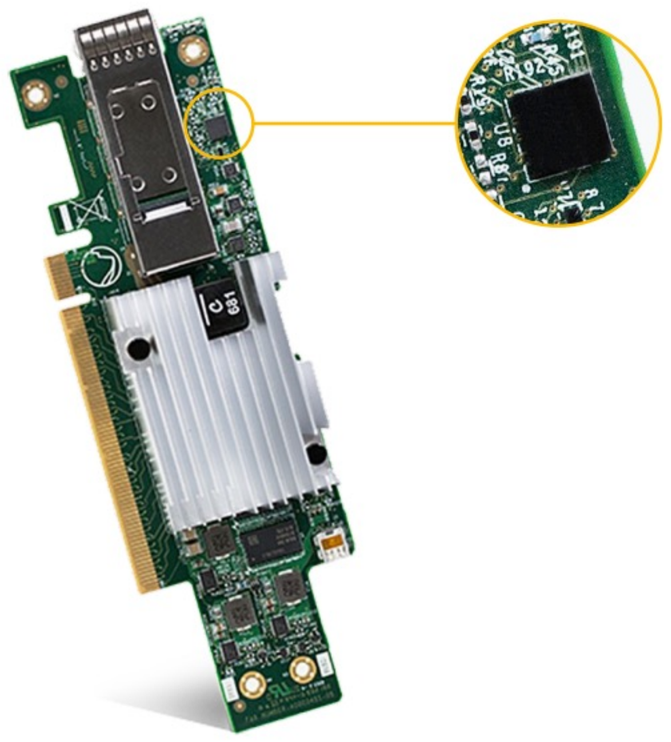
- Vulnerable to bus sniffing attacks
- TPM Reset attacks
  - SW reports hash values
- Bugs in the trusted software



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

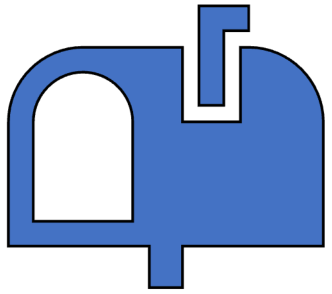




from https://www.hotchips.org/hc30/1conf/1.14_Google_Titan_GoogleFinalTitanHotChips2018.pdf

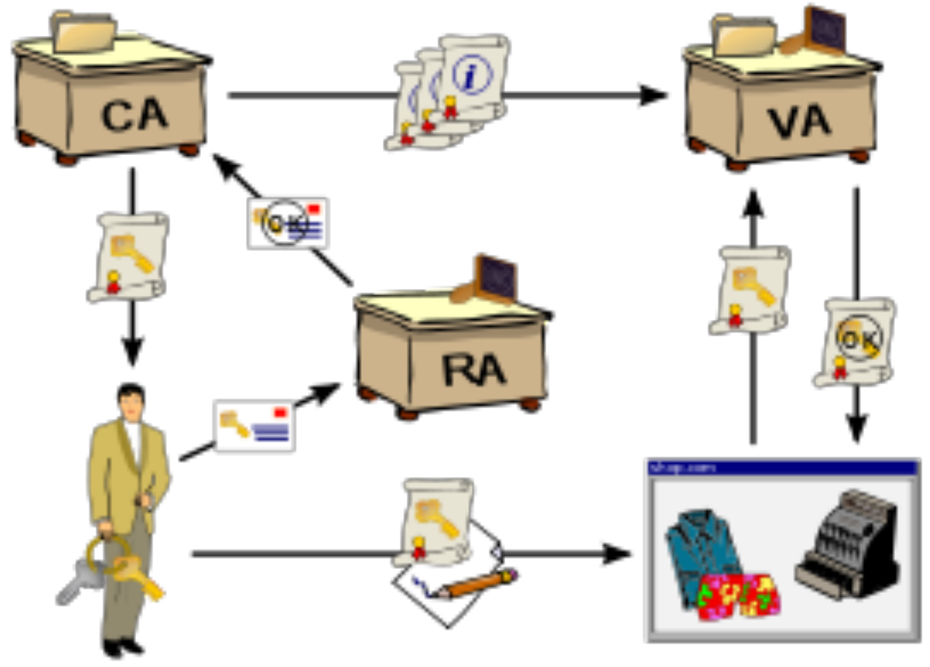
Identification – Public Key Cryptography (e.g., RSA, EC)

- A pair of keys:
  - Private key (K_{pri} – kept as secret); Public key (K_{pub} – safe to release publicly)
- Encryption:
  - Encrypt(plaintext, K_{pub}) = ciphertext
  - Decrypt(ciphertext, K_{pri}) = plaintext
- Digital signatures:
  - Proof that msg comes from *whoever owns private key corresponding to K_{pub}*
  - Sign(msg):
    - h = Hash(msg); signature = Encrypt(h, K_{pri})
    - Return {signature, msg}
  - Verify:
    - Decrypt(signature, K_{pub}) ?= Hash(msg)


Mail box is public;
Box key is private

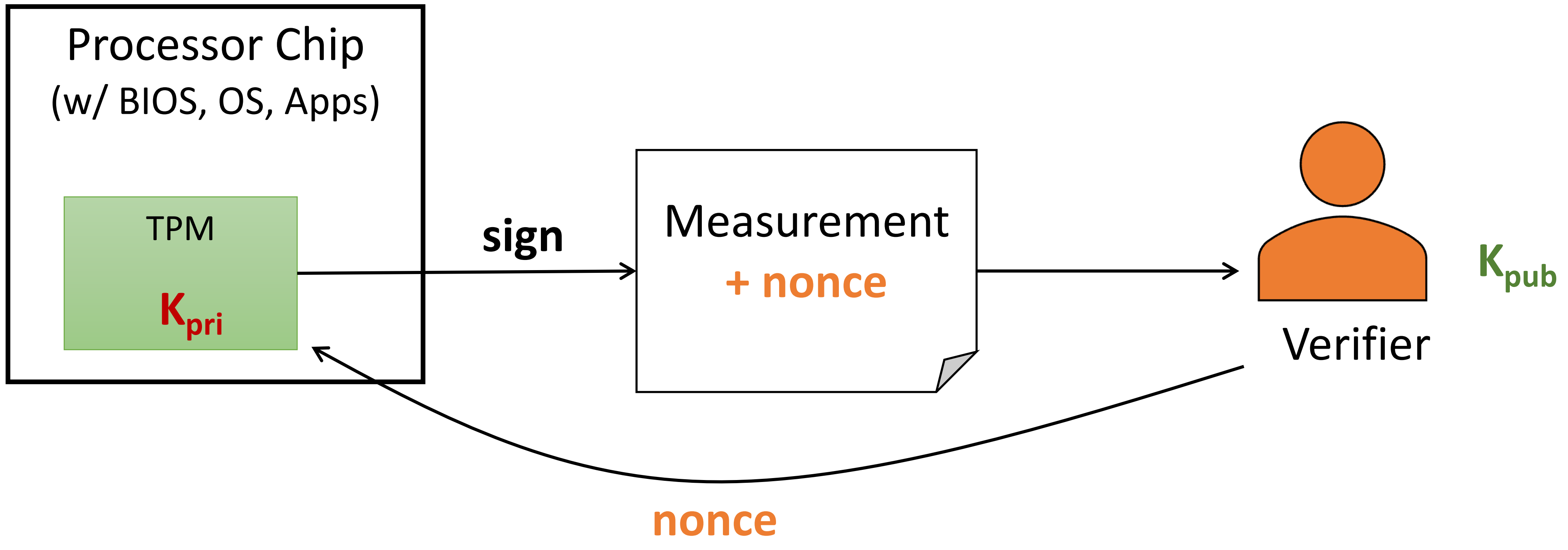
Public key infrastructures (PKIs)

- Bind public keys with identities
  -> website, chip



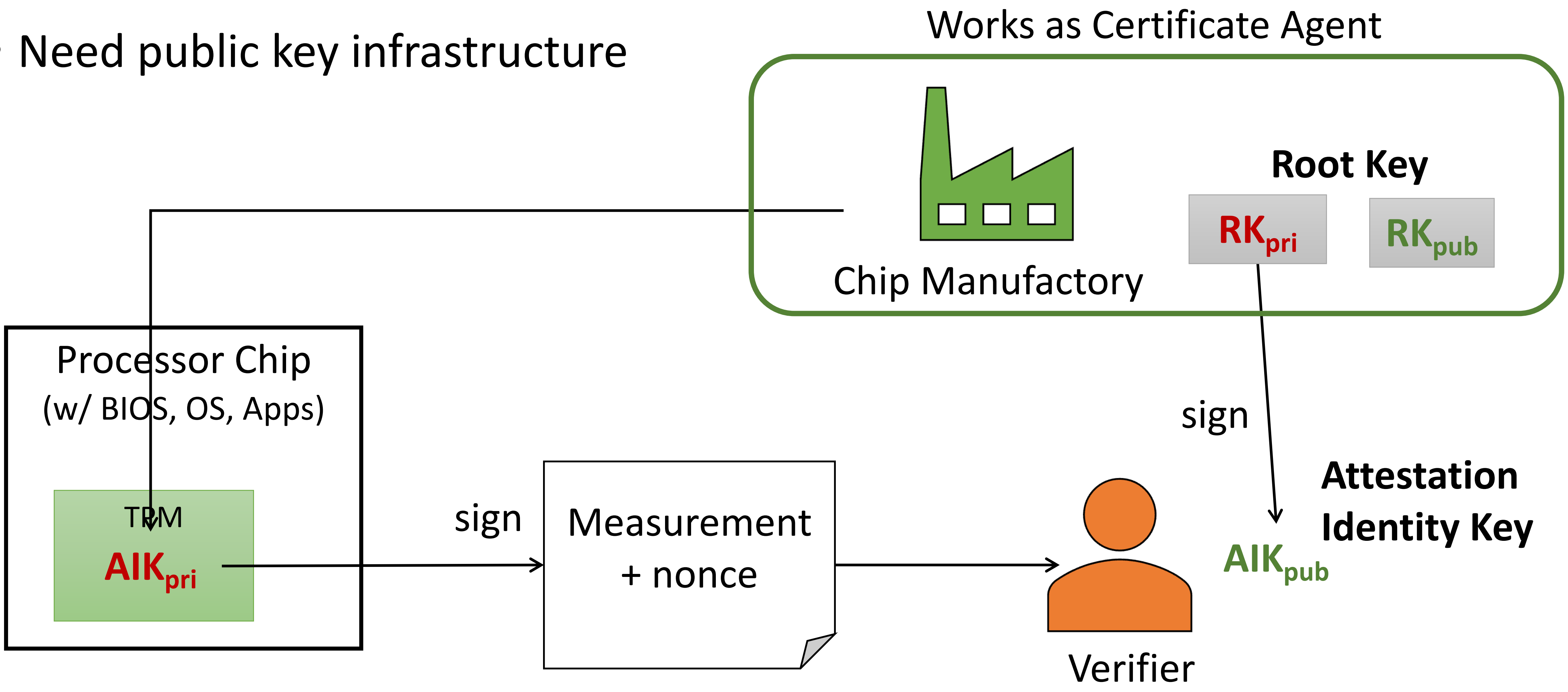
Platform Attestation

- Defend against replay attack: Freshness



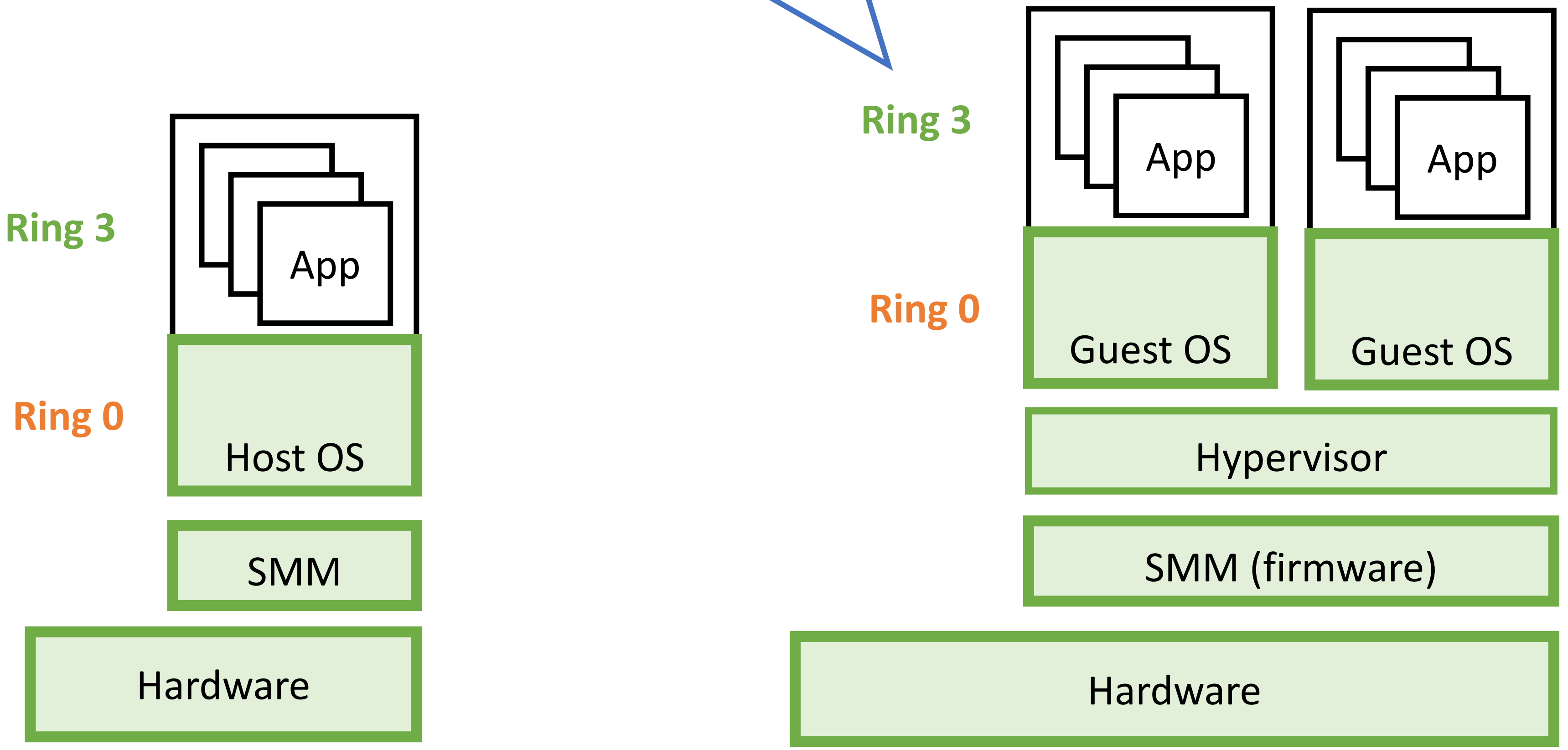
Platform Attestation

- Need public key infrastructure

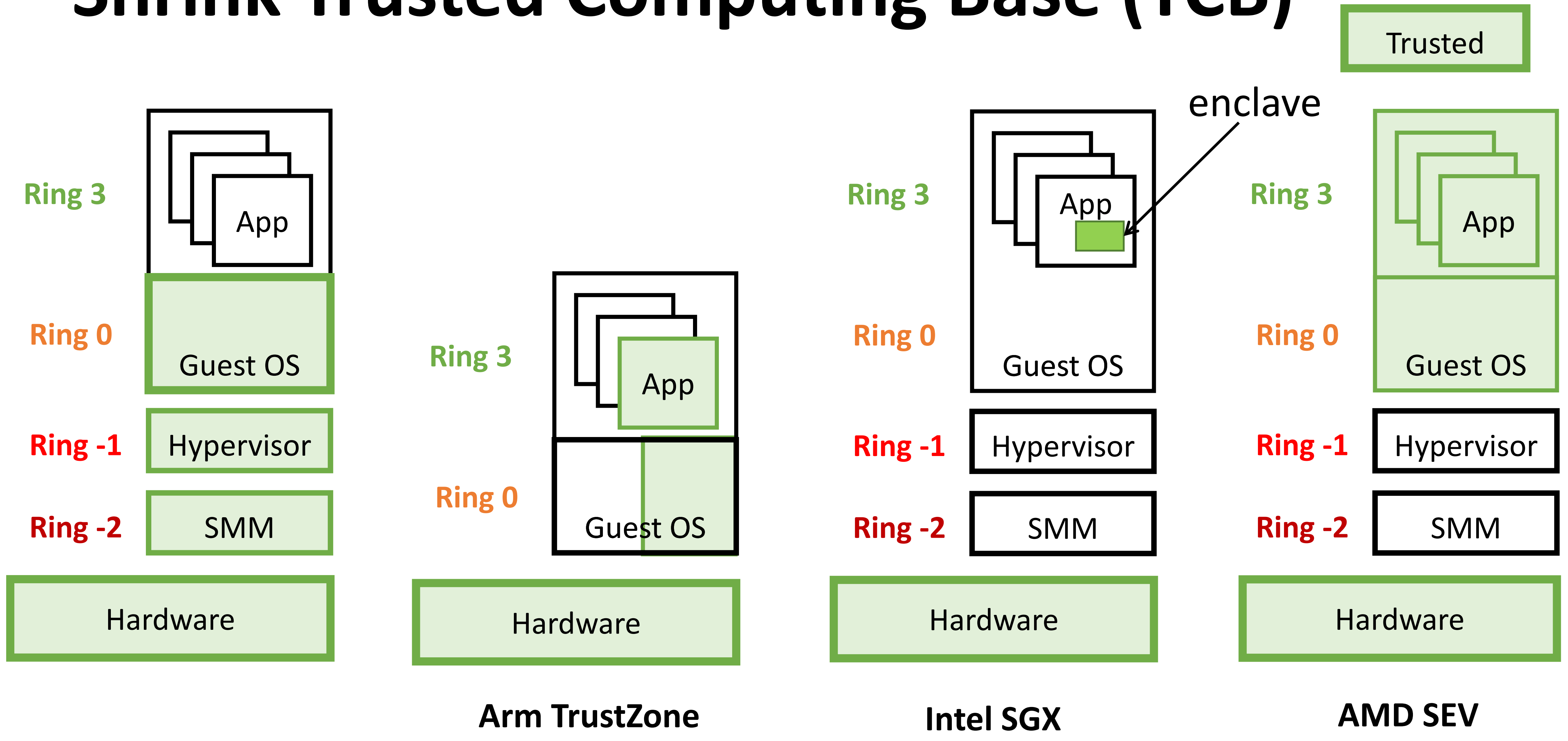


So Far ……

The trend: shrink TCB. Why?



Shrink Trusted Computing Base (TCB)



Next Lecture:
Side Channel Introduction