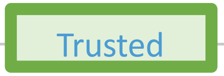
Comp 790-184: Hardware Security and Side-Channels

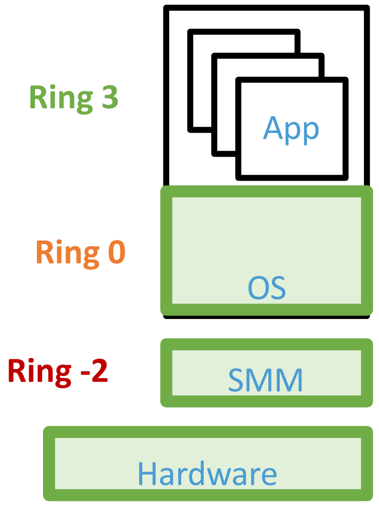
Lecture 7: Hardware-Supported Trusted Execution Environments (TEE)

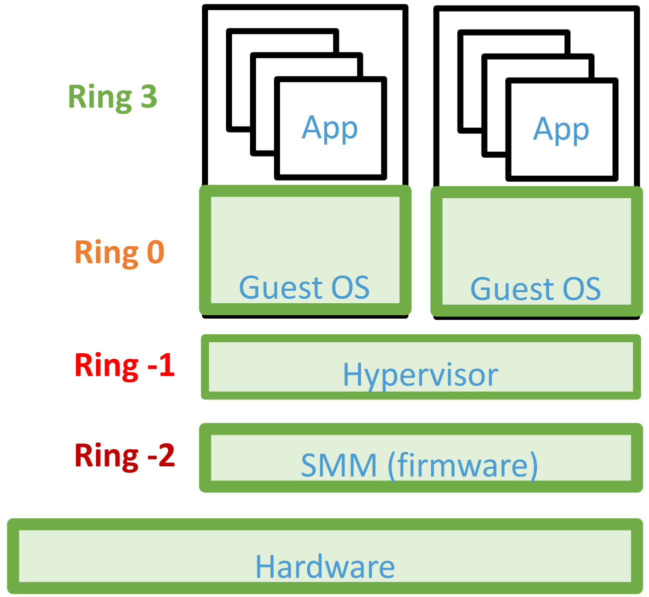
April 3, 2024
Andrew Kwong

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

Trusted Computing Base (TCB)



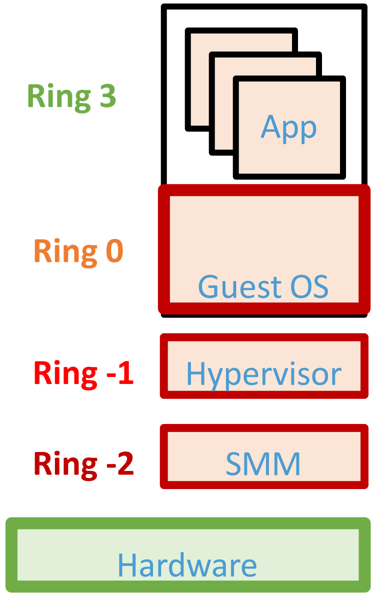




Shrink TCB. Why?

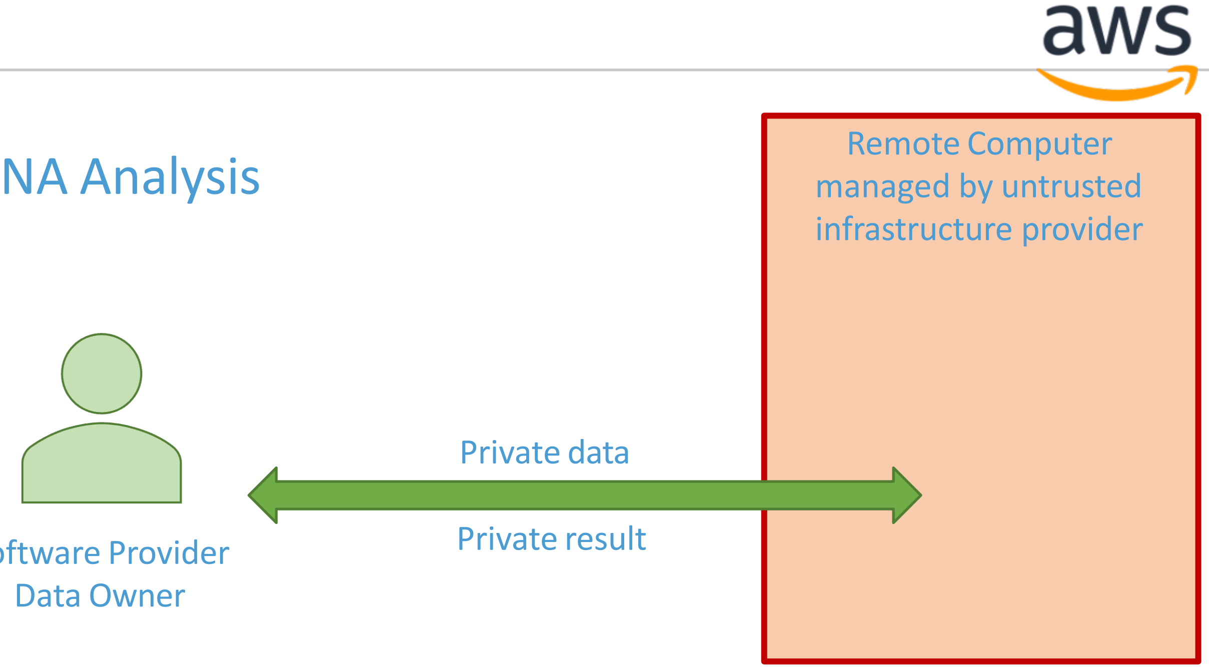
- Software bugs
  - SMM-based rootkits
  - Xen 150K LOC, 40+ vulnerabilities per year
  - Monolithic kernel, e.g., Linux, 17M LOC, 100+ vulnerabilities per year

  - Remote Computing
    - Remote computer and software stack owned by an untrusted party



Secure Remote Computing

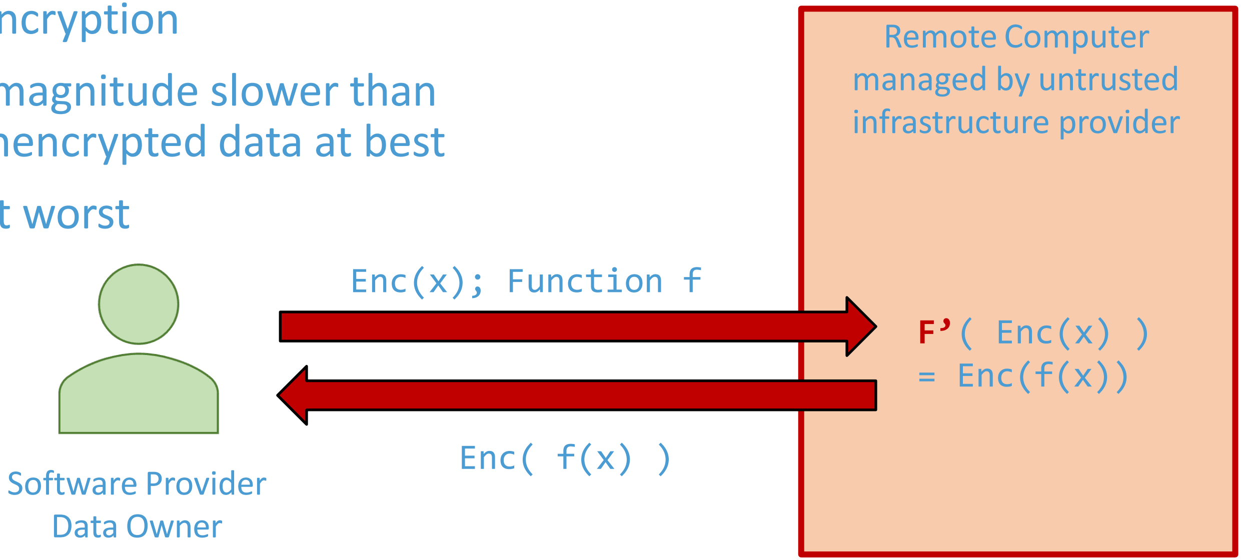
- Example: DNA Analysis


aws
Remote Computer managed by untrusted infrastructure provider
Private data
Private result
Software Provider
Data Owner

How can I keep my data private without trusting the host OS/hypervisor/SMM?

Software Solution

- Homomorphic Encryption
- **4 to 5 orders** of magnitude slower than computing on unencrypted data at best
    - Infeasible at worst

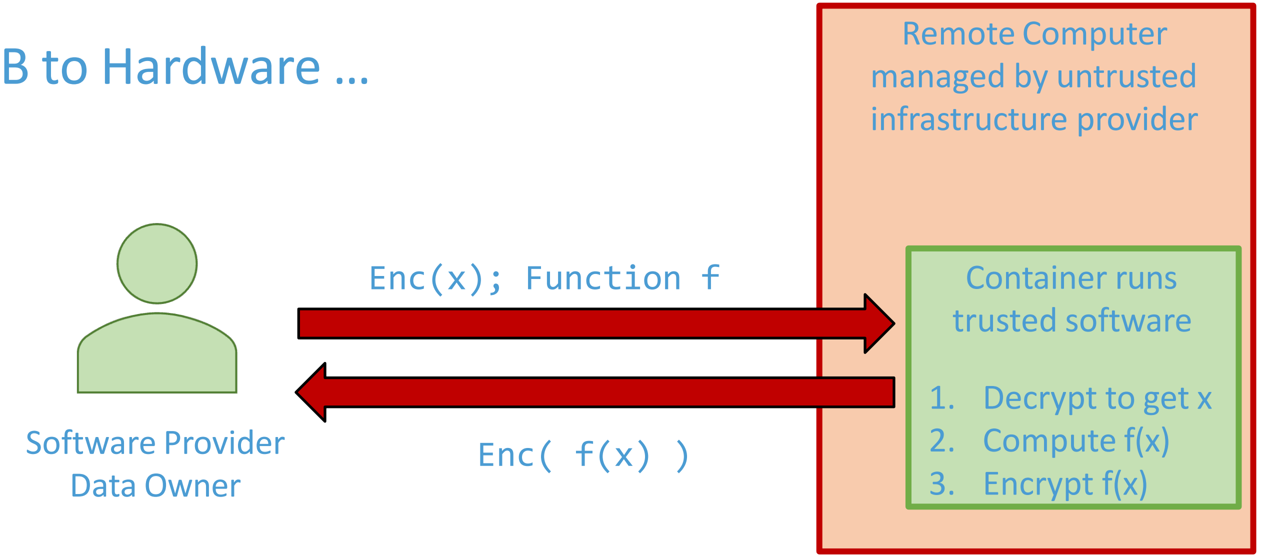

Remote Computer managed by untrusted infrastructure provider
Enc(x); Function f
F'(Enc(x)) = Enc(f(x))
Enc(f(x))
Software Provider Data Owner

- Performance? Accelerators?

  e.g., F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption; Axel Feldmann, Nikola Samardzic et al. MICRO'21

Hardware Solution

- Move TCB to Hardware …

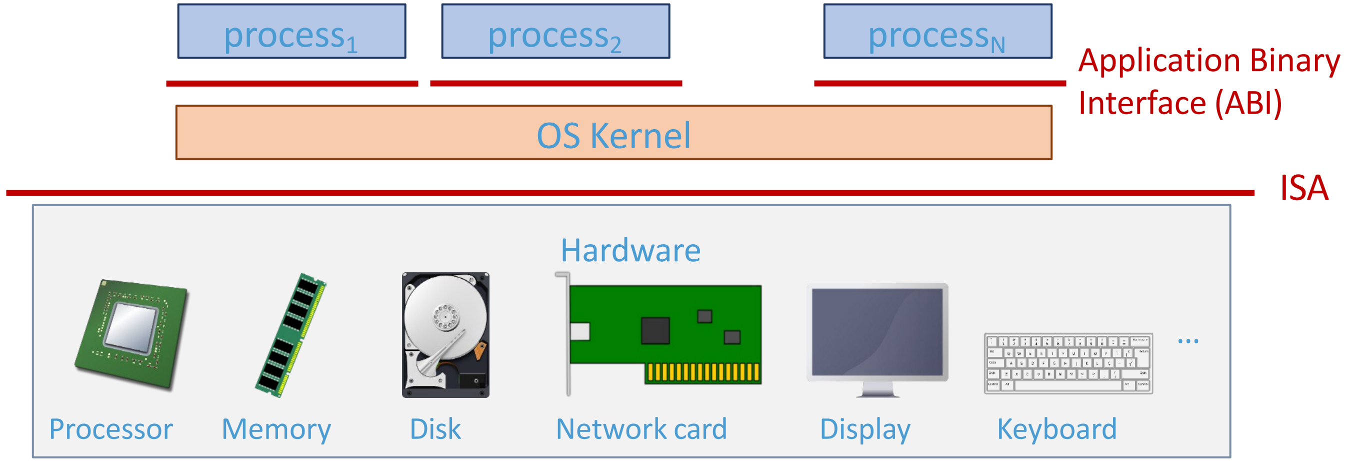


Outline

- Understand the threat model: privileged SW attacks
- Understand how to mitigate these threats

Privileged Software Attacks

Operating Systems



Launch Time

```
./helloworld
```

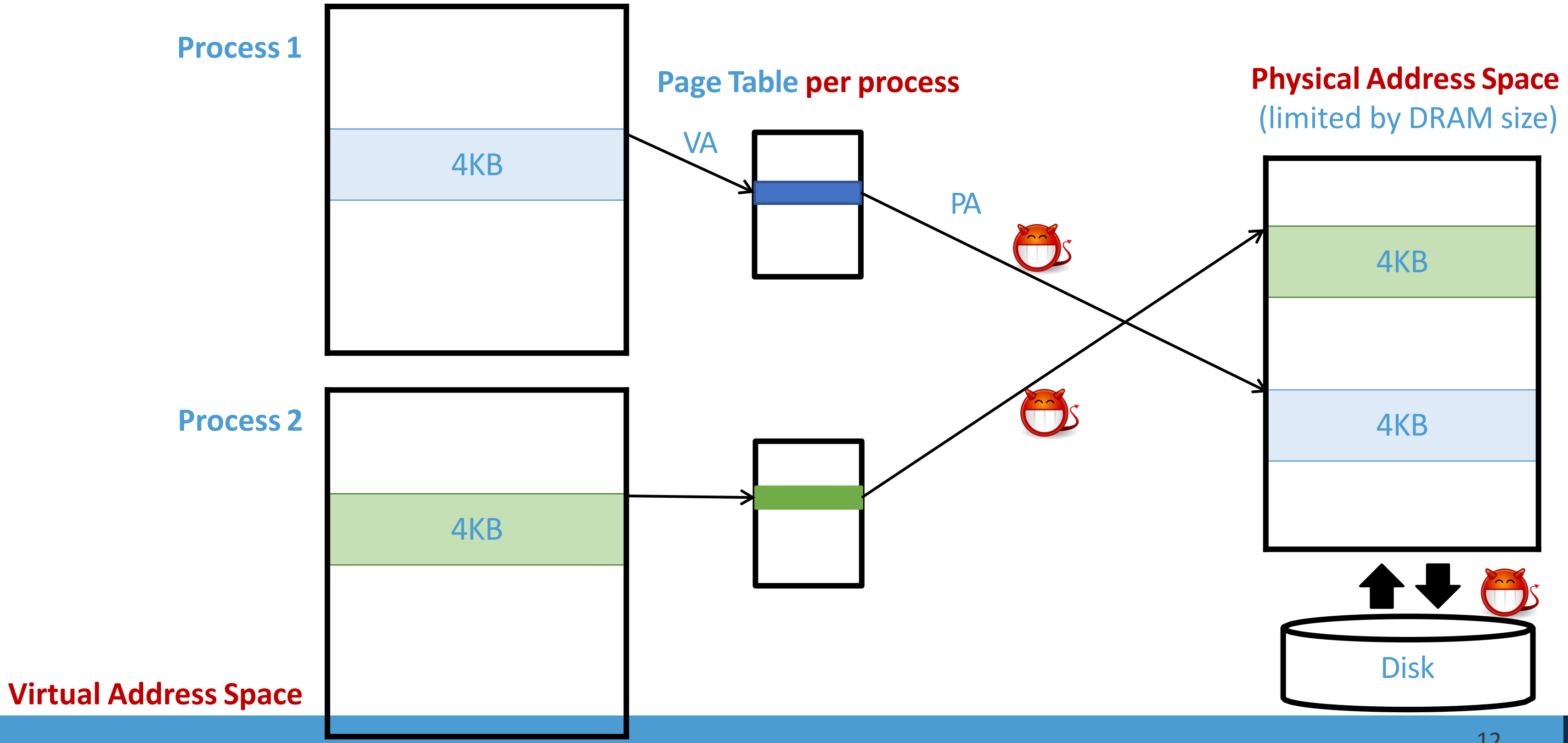
- Operations at launch time:
  - Create a process (PID, status, etc.)
  - Create a virtual address space: allocate memory for stack, heap, code region, set up the page tables
  - Setup file descriptor for input and output
  - Load the binary into the code region, and linked library if needed
  - Transfer the control to user space

CPU Abstraction

- Expose to users threads, rather than physical cores
- Achieve via context switch and interrupt handling

- Switch from user space to kernel space
  - Remember the current PC
  - Jump to kernel code: perform a sequence of save operations
    - Save general purpose registers content into an object associated with the current thread
    - Save system registers, including page table root address (CR3 in X86)
  - Based on the interrupt type, decide what to do
- Switch back to user space
  - Restore all the registers: general-purpose + system registers
  - Jump back to the saved PC

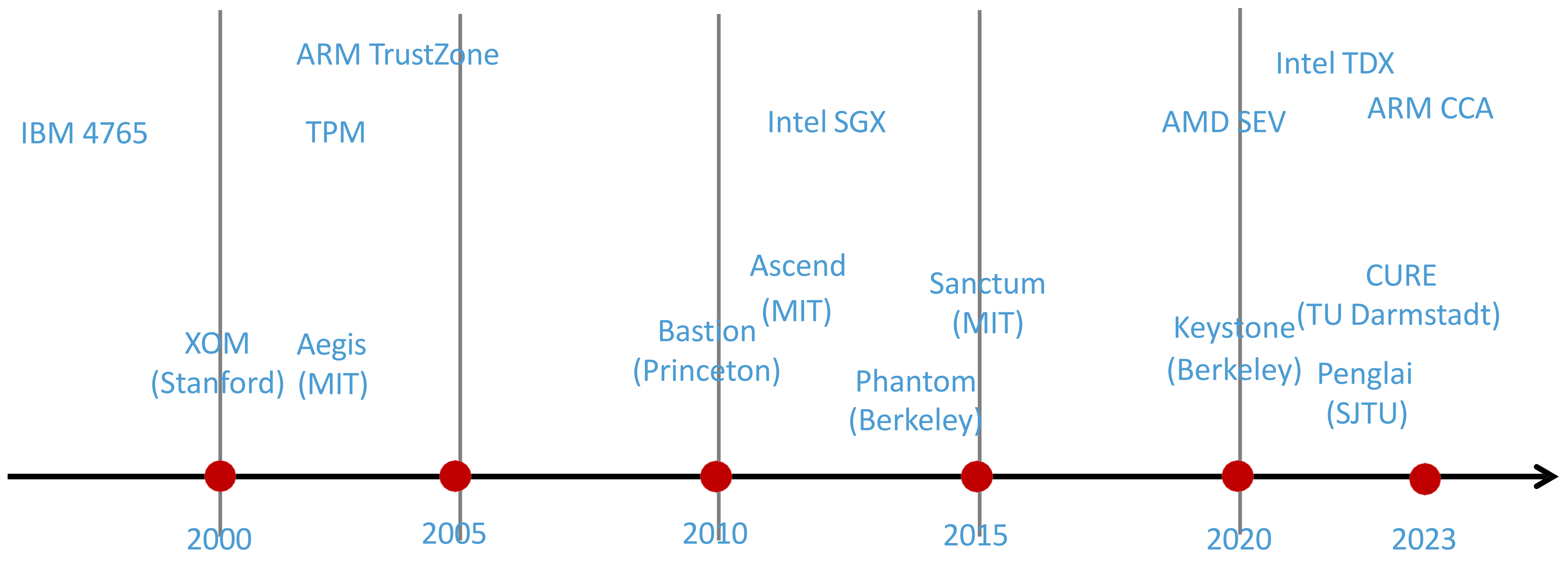
Virtual Memory Abstraction



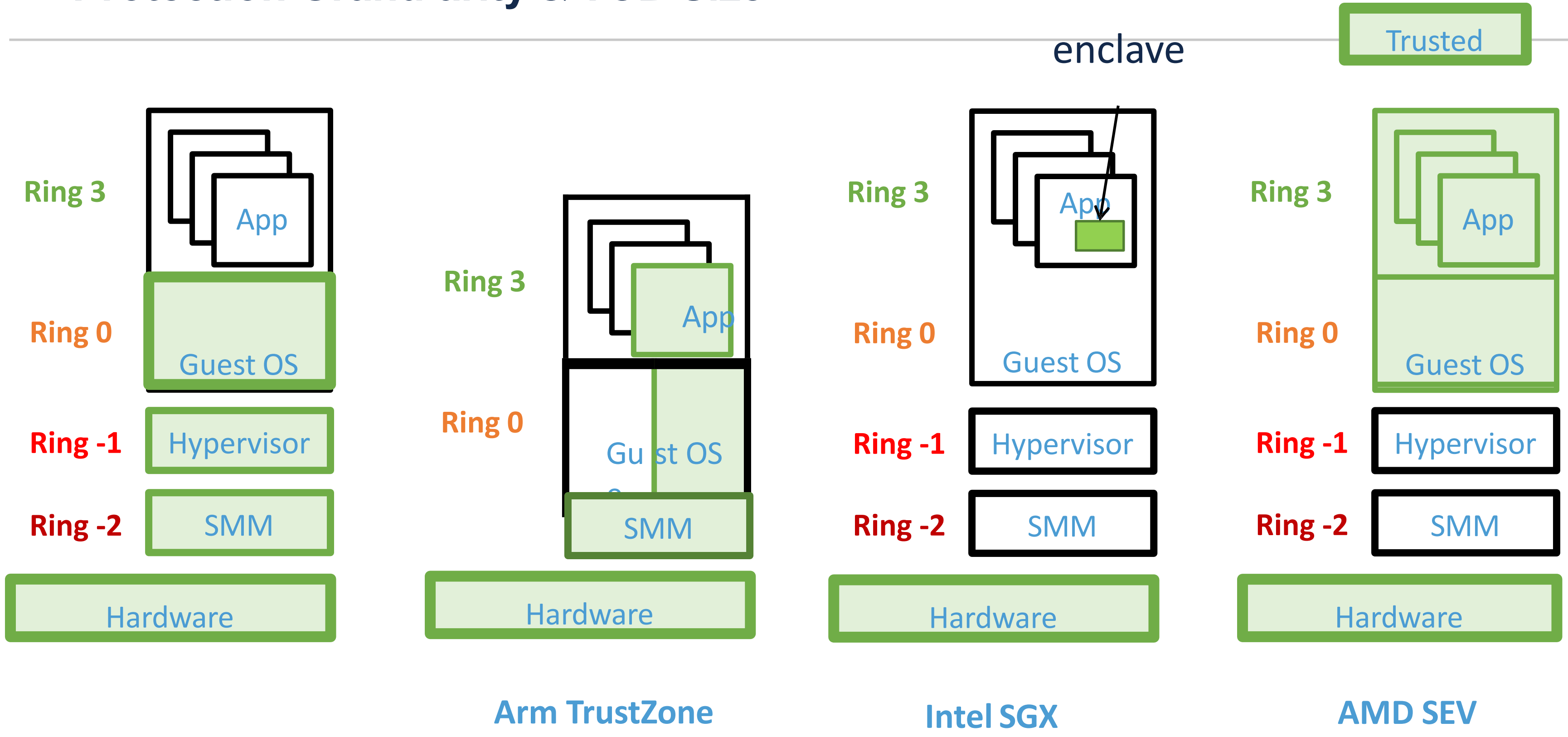
What can a privileged software attacker do?

- A non-comprehensive list
  - Modify the code to be executed
  - Monitor the whole execution process and data in register and in memory
  - Modify data in register and memory
  - Intercept IO, eavesdrop and tamper with the communication
  - ……

TEE Examples

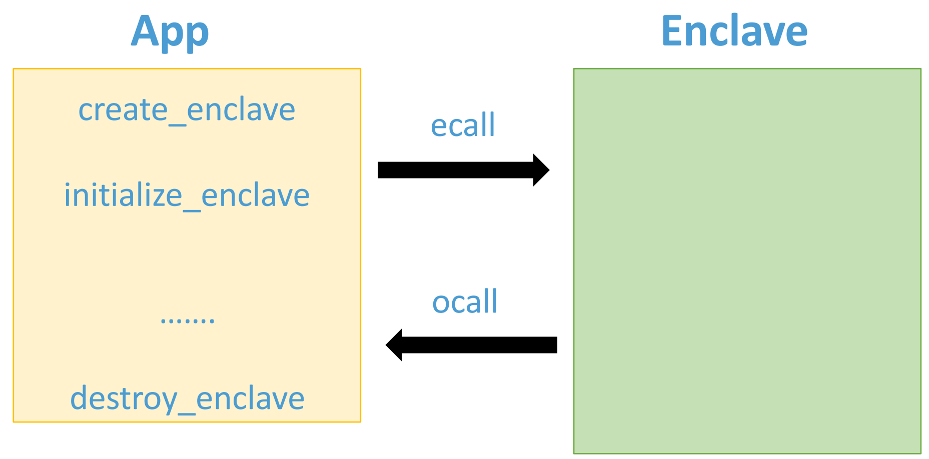


Protection Granularity & TCB Size



SGX Enclave Programming Model

- Examples from: https://github.com/intel/linux-sgx



Security Tasks

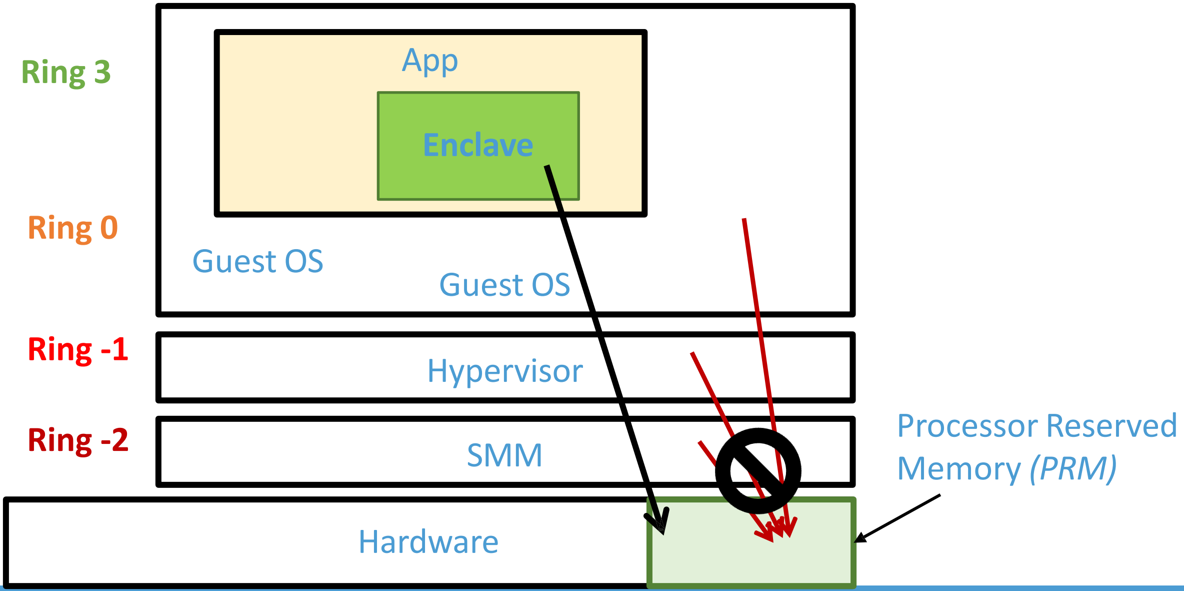
- How do we ensure the runtime execution follows our expectation (confidentiality and integrity of the execution)?

- How do we ensure the enclave code is the code that we want to execute? (code integrity during initialization)

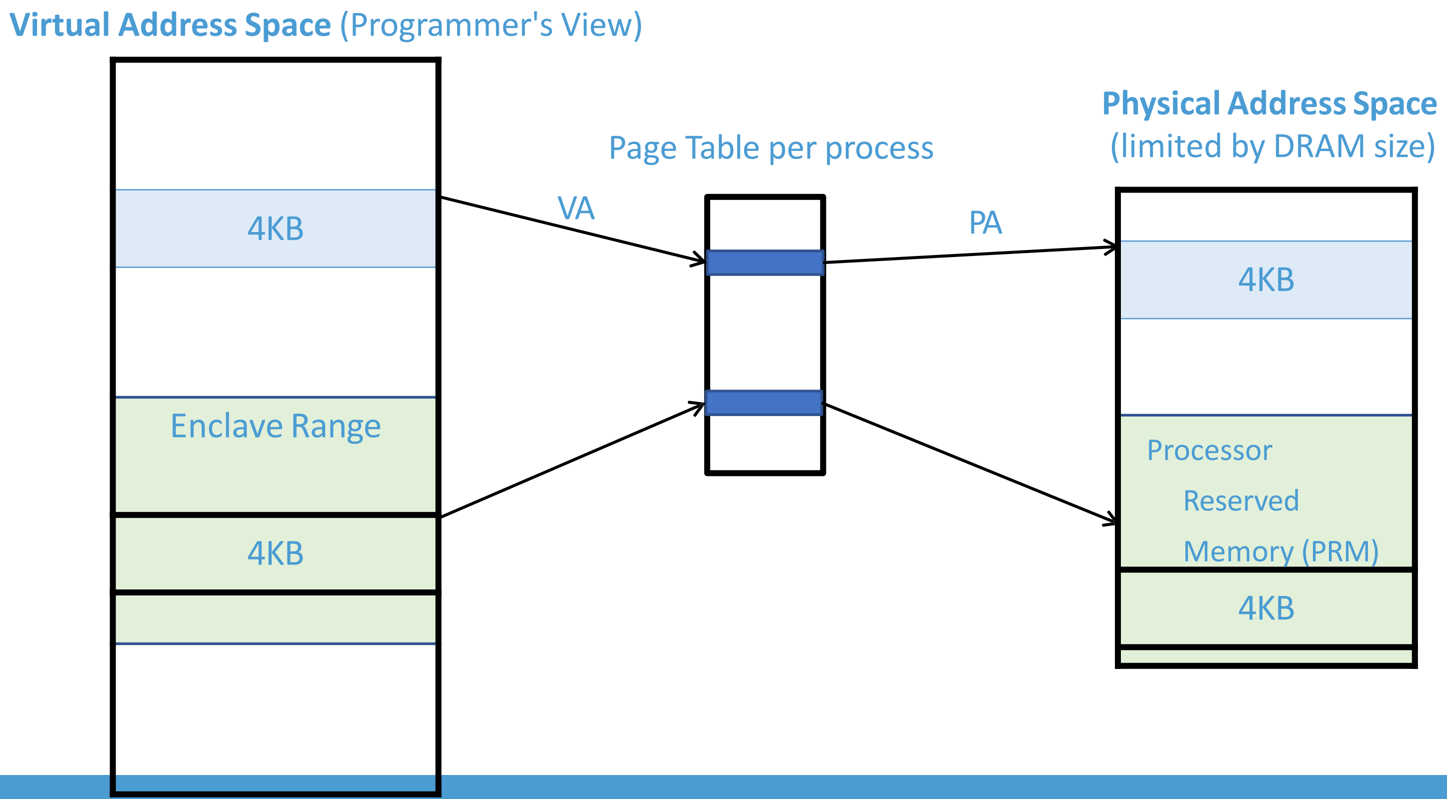
- DRAM security? How to deal with Rowhammer and Coldboot attacks?

Intel SGX Overview

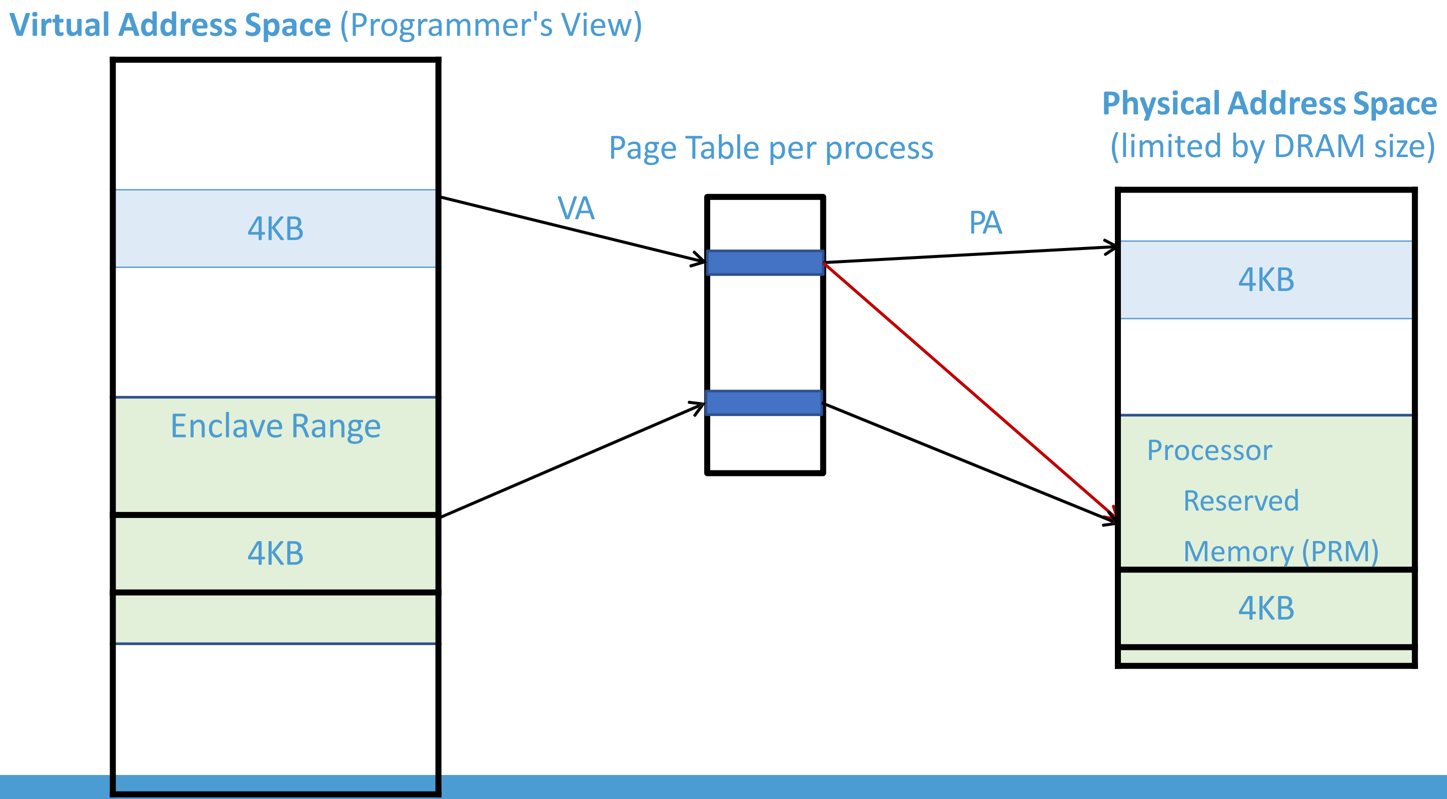
- Enclave code/data map to PRM; Different enclaves access their own memory region



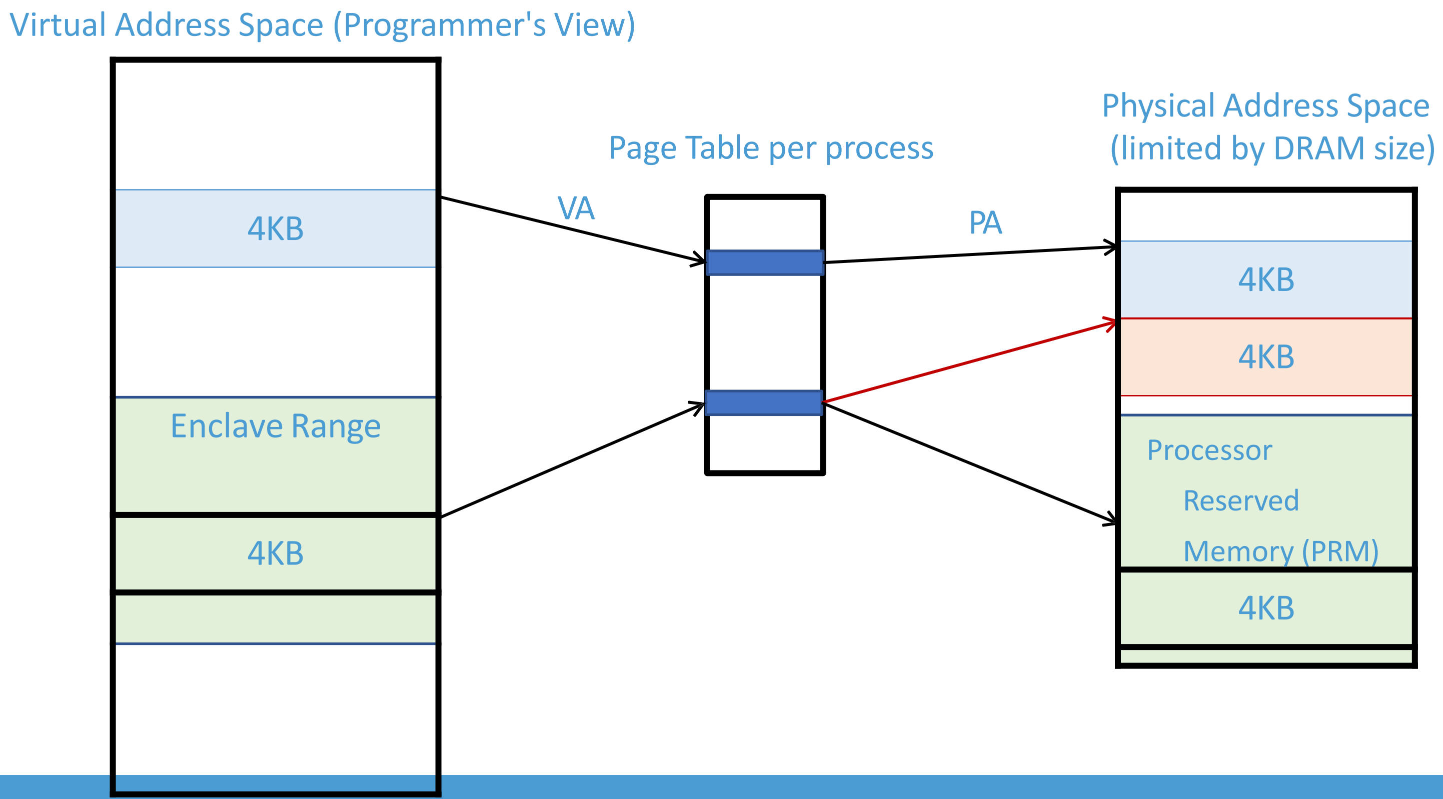
Intel SGX Address Translation Overview



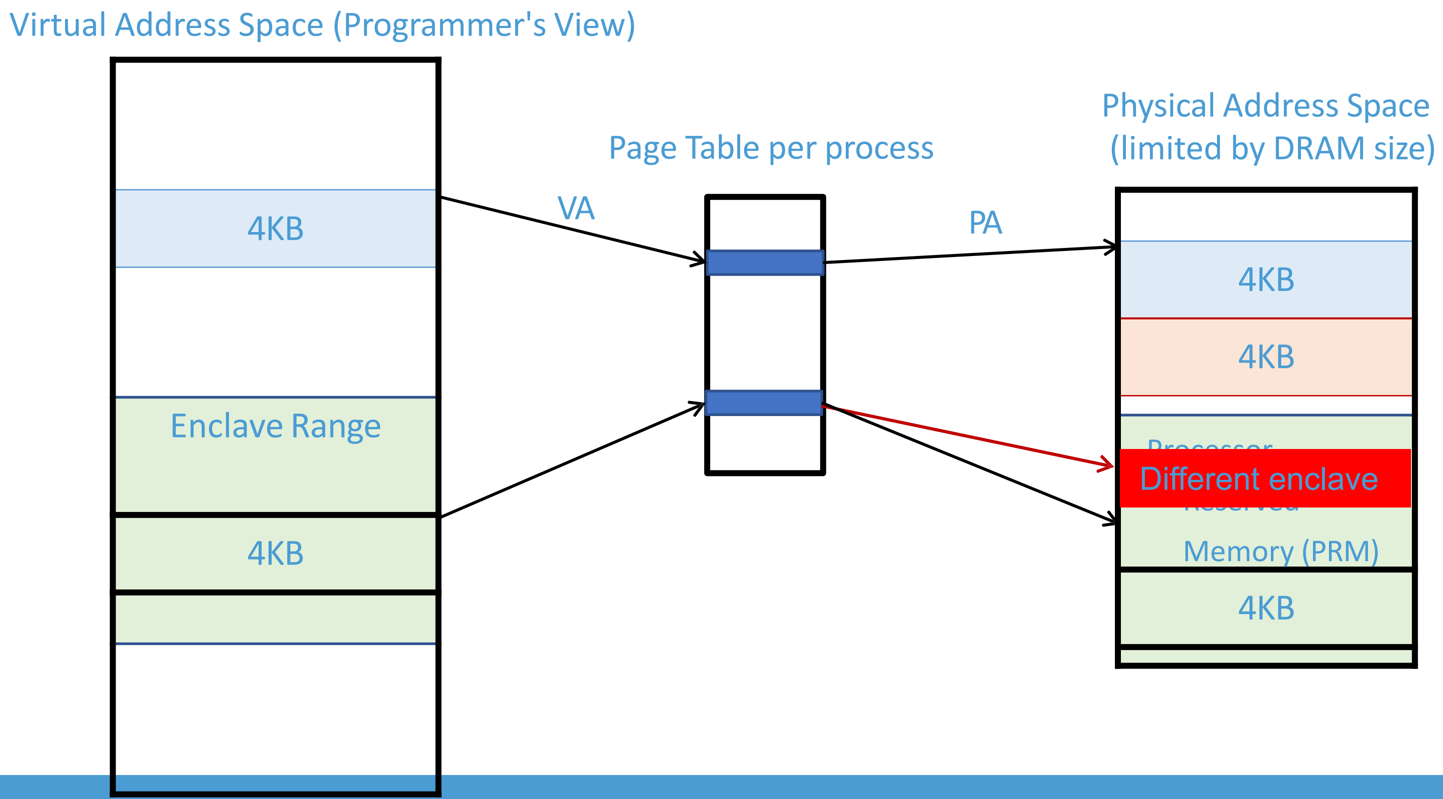
Malicious Address Translation #1



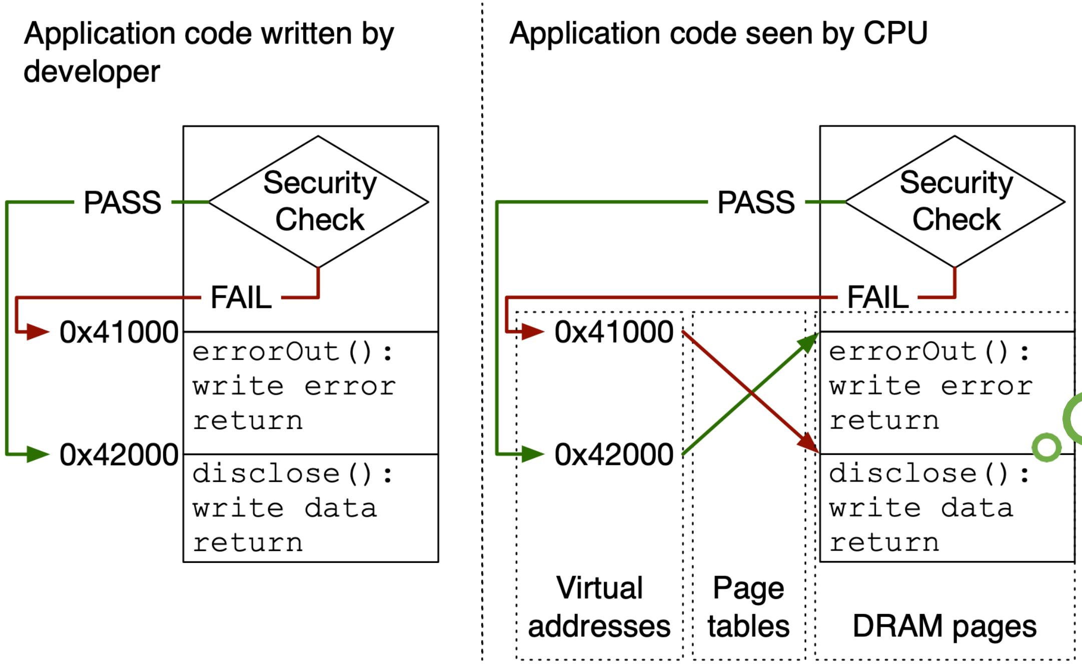
Malicious Address Translation #2



Malicious Address Translation #2



Malicious Address Translation #4



Need to keep track of the page table for enclaves by trusted hardware/software.

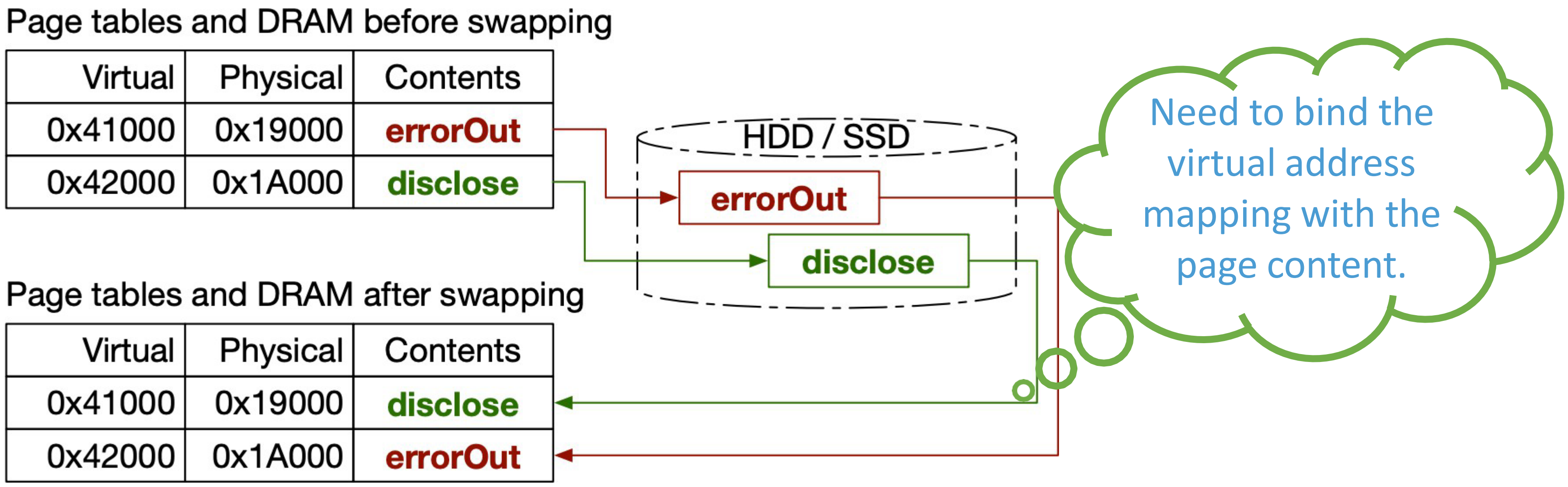
Solution: Inverted Page Table

- Check for security invariant:
  - Enclave VA, enclave mode ➔ PRM
  - Non-enclave mode is not allowed access to the PRM at all

- For each page in the PRM, remember the mapping from
  <PPN> ➔ <VPN, Enclave ID>
  Keep the reversed page table in PRM, so privilege software cannot modify

Malicious Address Translation #5

A memory mapping attack that does not require modifying the page tables.



Page tables and DRAM before swapping

Virtual	Physical	Contents
0x41000	0x19000	errorOut
0x42000	0x1A000	disclose

Page tables and DRAM after swapping

Virtual	Physical	Contents
0x41000	0x19000	disclose
0x42000	0x1A000	errorOut

Need to bind the virtual address mapping with the page content.

Solution: Page Encryption and Authentication

- Pages are encrypted under different keys
- MAC over enclave ID, VPN, unique key, data, nonce
    - Need to prevent replay attacks

Summary: SGX Memory Management

- Untrusted OS handles paging and swapping
- Maintain an inverted page table and check after every address translation

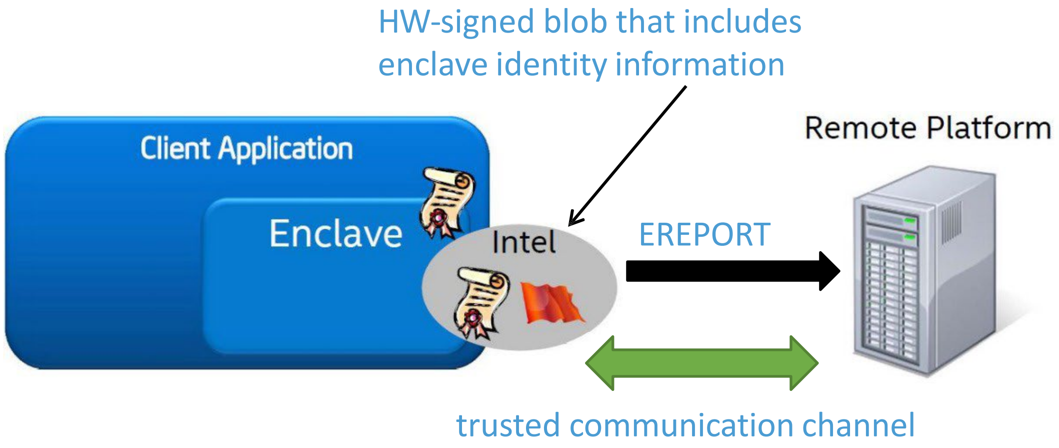
  Physical page in PRM -> (enclave ID, virtual page number)

- Encrypt/decrypt upon page swap to non-PRM region

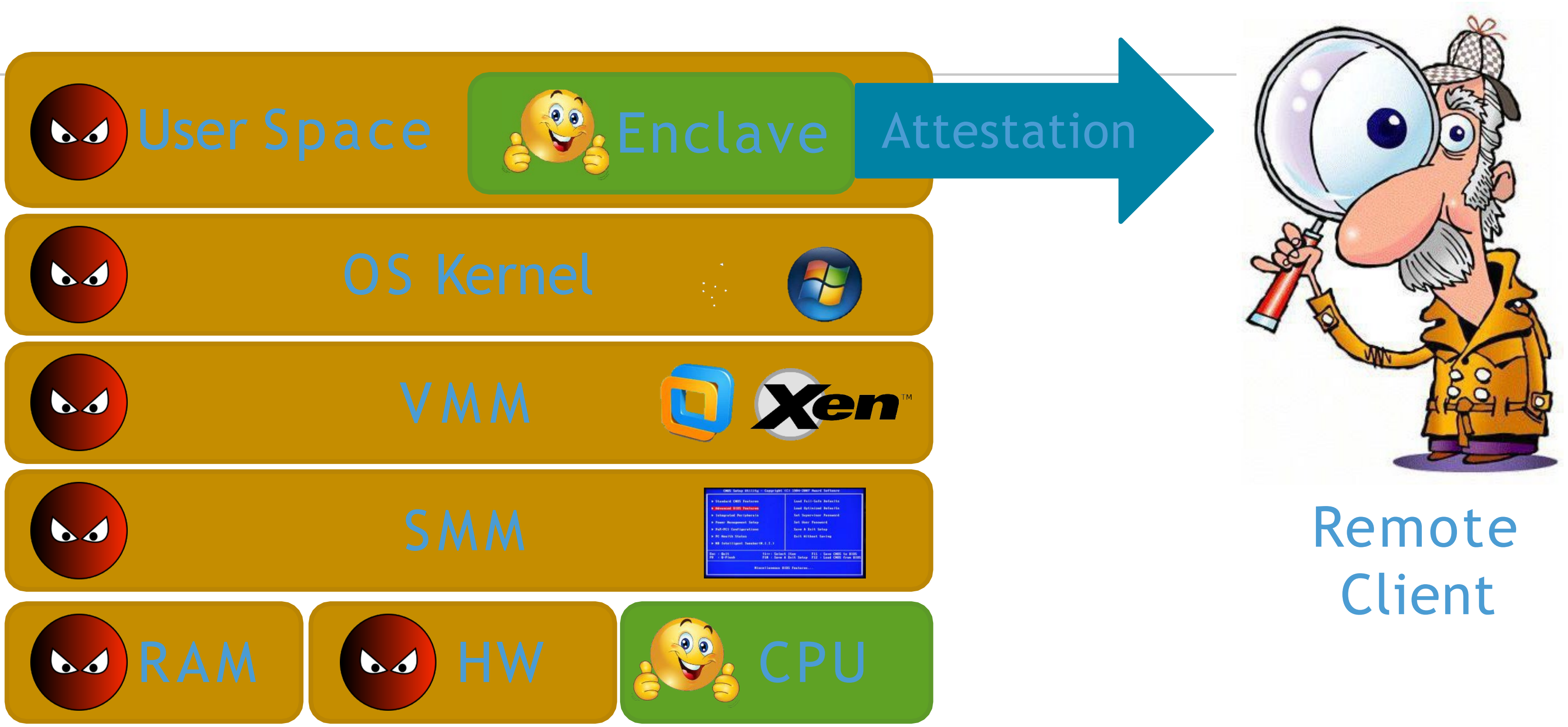
  (nonce, enclave ID, virtual page number, key, page content) ➔ MAC

Remote Attestation

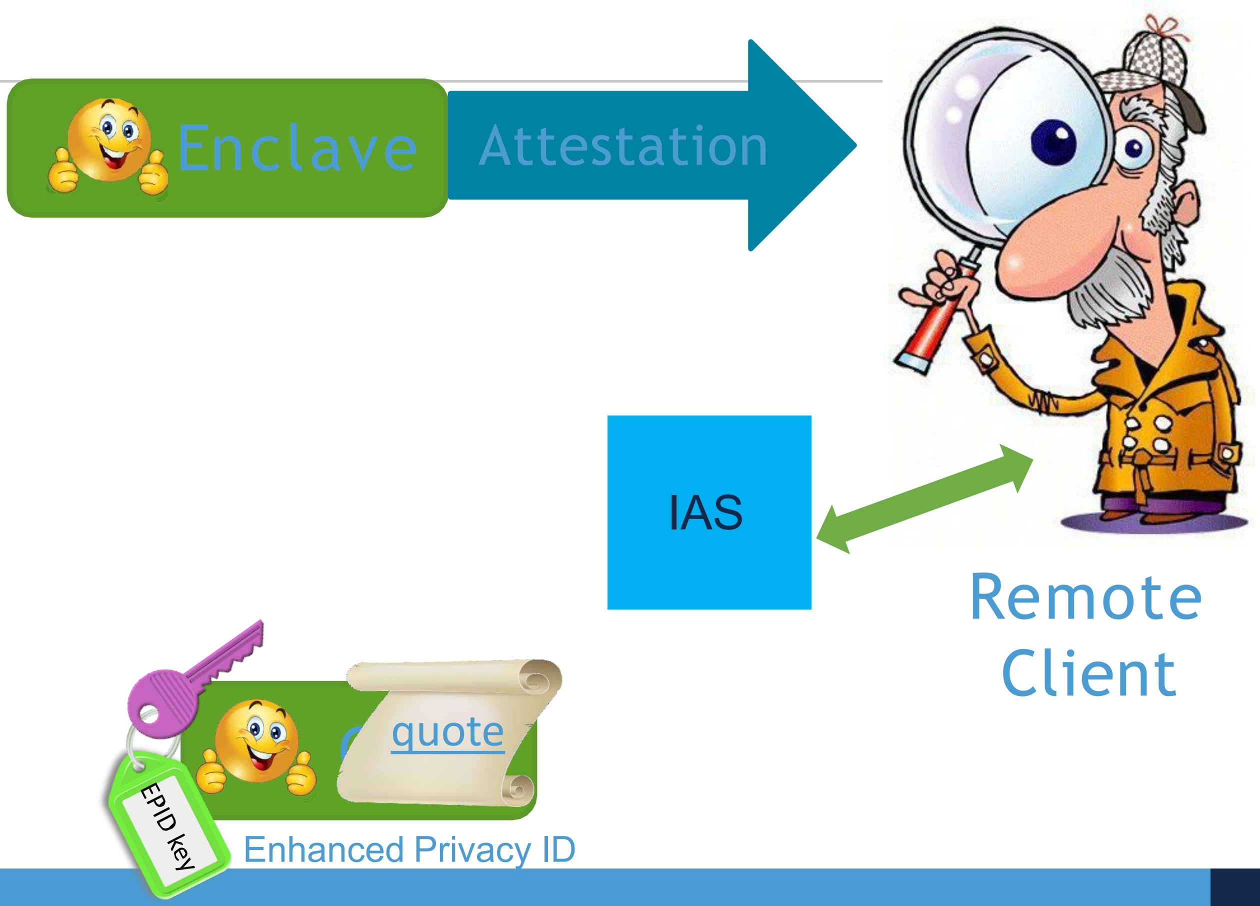
- HW based attestation provides proof that “this is the right application executing on an authentic platform” (approach similar to secure boot attestation)


HW-signed blob that includes enclave identity information
Client Application
Enclave
Intel
EREPORT
Remote Platform
trusted communication channel

Software Guard Extensions (SGX) Security Model



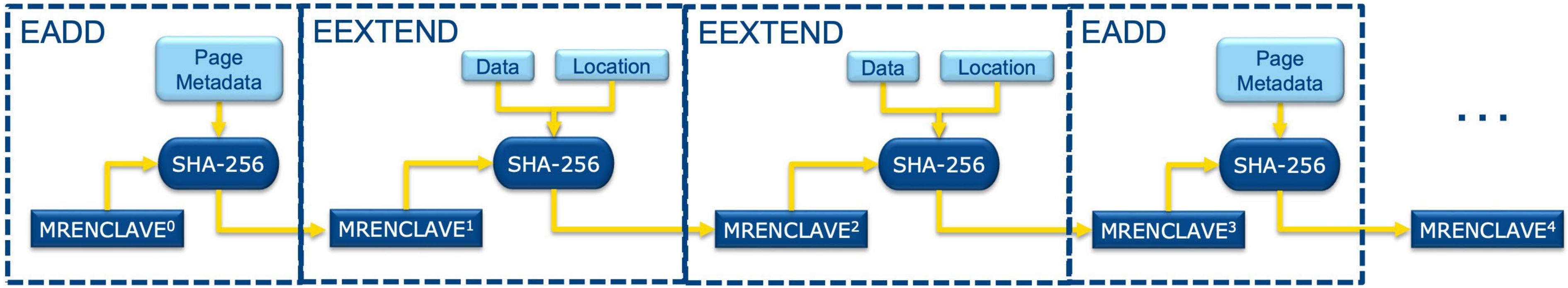
Software Guard Extensions (SGX) Security Model



Takeaway: trust is based on the EPID key

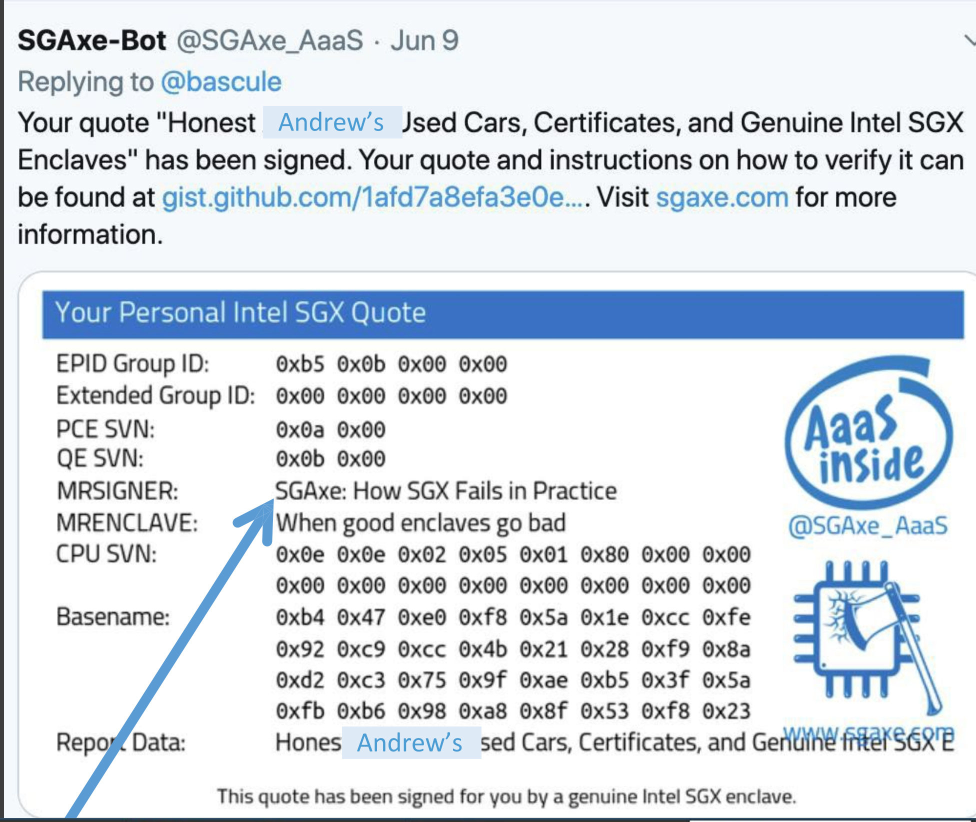
Enclave Measurement

- Hardware generates a cryptographic log of the build process
  - Code, data, stack, and heap contents
  - Location of each page within the enclave
  - Security attributes and enclave capabilities
  - Firmware version, hyperthreading
- Enclave identity (MRENCLAVE) is a 256-bit digest of the log that represents the enclave



AaaS (Attestation as a Service)

- @SGAxe_AaaS
  Will attest to anything tweeted at it
- Signed 100+ quotes within 2 hours
  - Blocked by Github
- After the public release of the paper, key was still trusted for a whole month
- Can't update TCB quickly because SGX users need to install BIOS updates
- Hardcoded MRSIGNER prevents abuse



SGAxe-Bot @SGAxe_AaaS · Jun 9

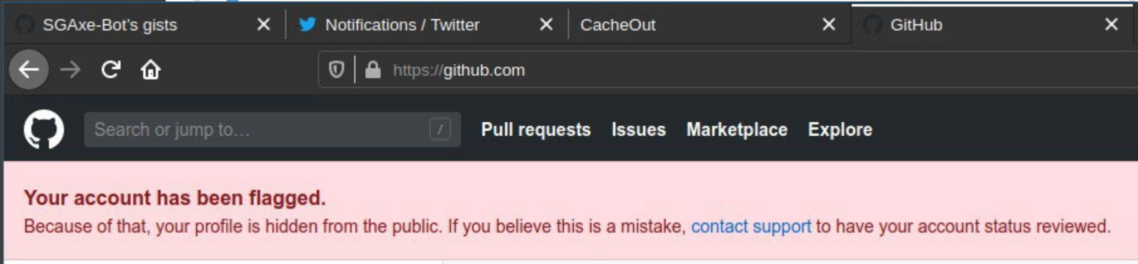
Replying to @bascule

Your quote "Honest Andrew's Jsed Cars, Certificates, and Genuine Intel SGX Enclaves" has been signed. Your quote and instructions on how to verify it can be found at gist.github.com/1afd7a8efa3e0e…. Visit sgaxe.com for more information.

Your Personal Intel SGX Quote

EPID Group ID:	0xb5 0x0b 0x00 0x00
Extended Group ID:	0x00 0x00 0x00 0x00
PCE SVN:	0x0a 0x00
QE SVN:	0x0b 0x00
MRSIGNER:	SGAxe: How SGX Fails in Practice
MRENCLAVE:	When good enclaves go bad
CPU SVN:	0x0e 0x0e 0x02 0x05 0x01 0x80 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
Basename:	0xb4 0x47 0xe0 0xf8 0x5a 0x1e 0xcc 0xfe 0x92 0xc9 0xcc 0x4b 0x21 0x28 0xf9 0x8a 0xd2 0xc3 0x75 0x9f 0xae 0xb5 0x3f 0x5a 0xfb 0xb6 0x98 0xa8 0x8f 0x53 0xf8 0x23
Report Data:	Hones Andrew's sed Cars, Certificates, and Genuine Intel SGX E

This quote has been signed for you by a genuine Intel SGX enclave.

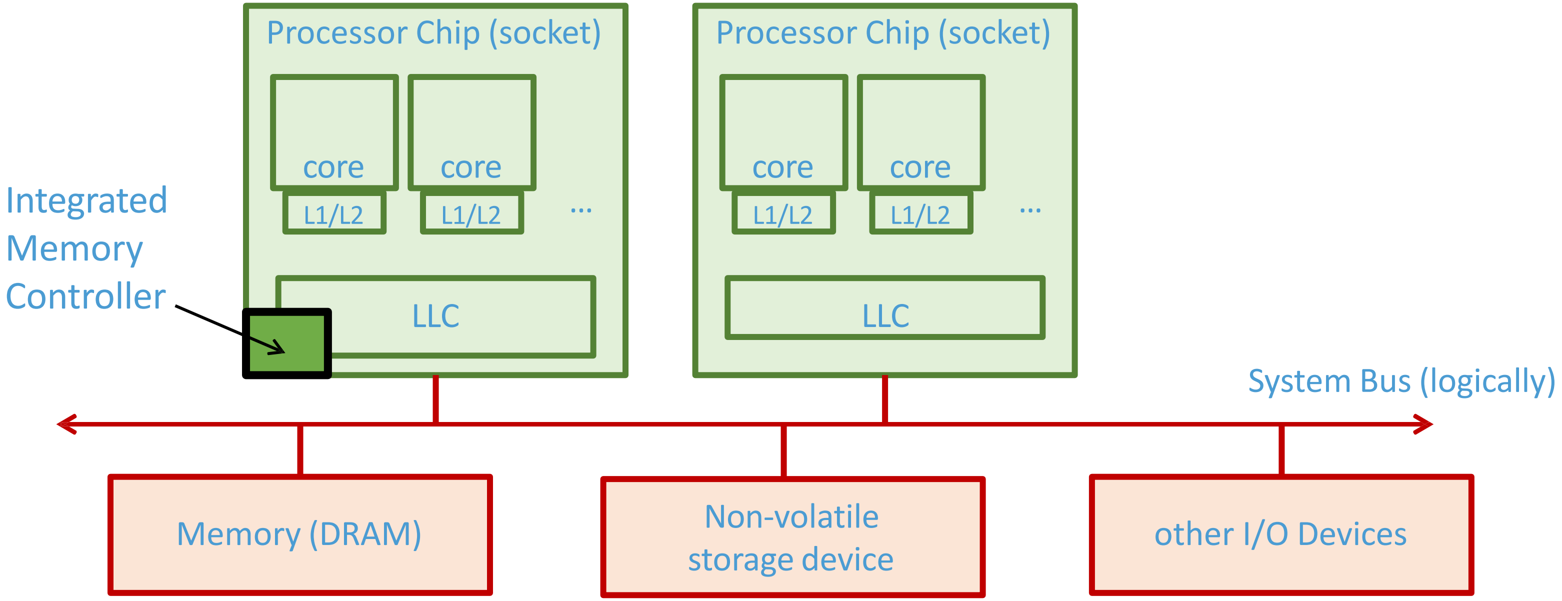


Your account has been flagged.

Because of that, your profile is hidden from the public. If you believe this is a mistake, contact support to have your account status reviewed.

Additional Security Threats

- DRAM attacks: Rowhammer, Coldboot attacks

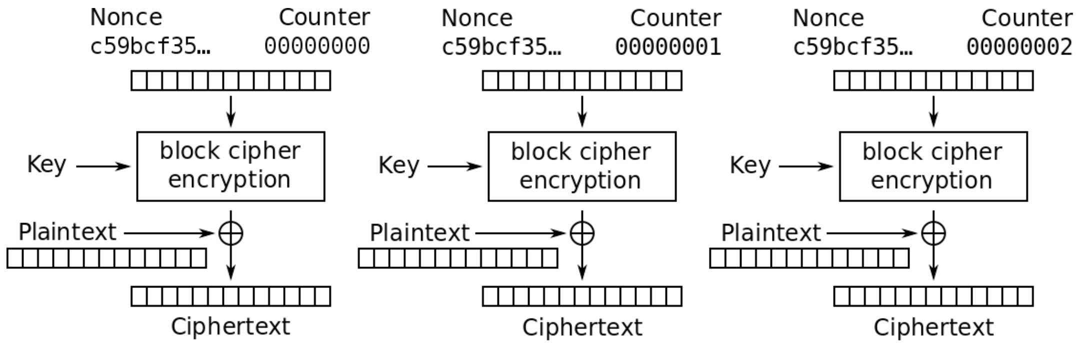


Memory Encryption Engine (MEE)

- Confidentiality:
  - DATA written to the DRAM cannot be distinguished from random data.
- Integrity + freshness:
  - DATA read back from DRAM to LLC is the same DATA that was most recently written from LLC to DRAM.
- *What attacks can be mitigated?*
  - *Rowhammer?*
  - Bus-tapping?
  - Cold-boot?
  - Side-channels on memory accesses?

Confidentiality

- AES 128-CTR mode

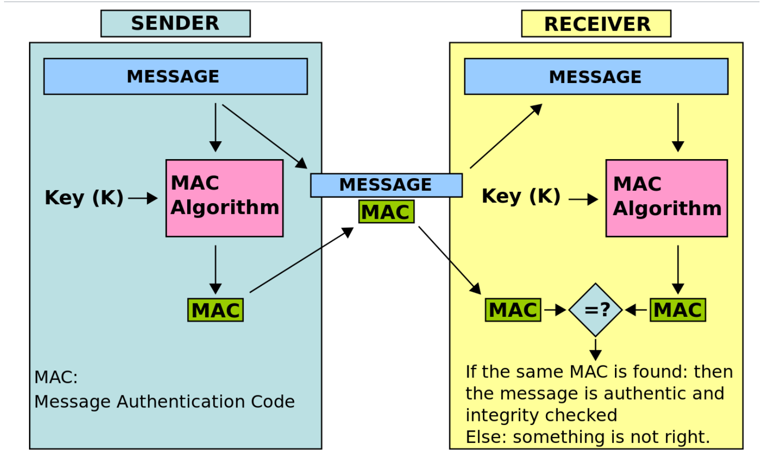


Counter (CTR) mode encryption

Message Authentication Code (MAC)

- MAC provides integrity and authentication

- Freshness
  - `hash = SHA(message || nonce)`
  - `HMAC = enc(hash, key)`

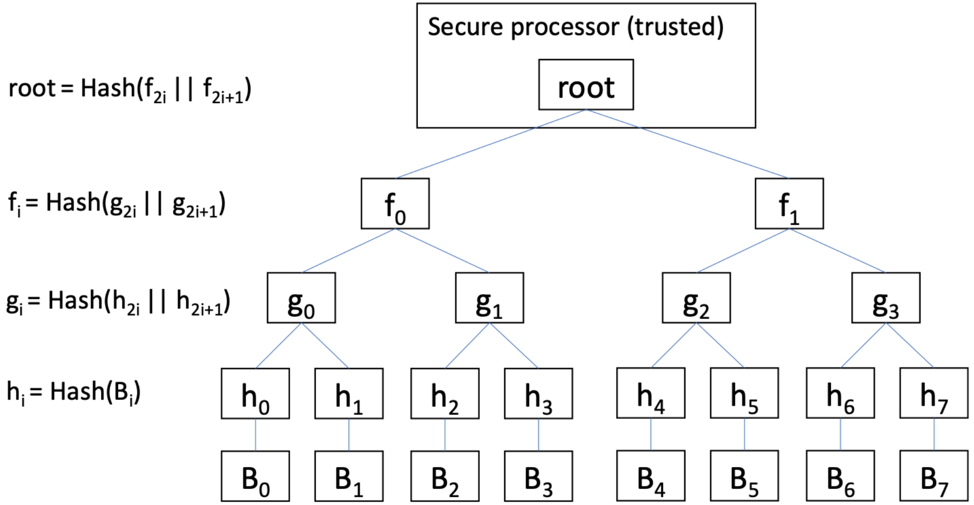

SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MAC:
Message Authentication Code
MESSAGE
MAC
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

Integrity Storage Problem

- For each cache line: {ciphertext + CTR + MAC}
  - MAC 56 bits
  - CTR 56 bits
- Can we store all the three components off-chip?
- Problem: if store CTR on-chip ➔ high on-chip storage requirement

Operations on Merkle Tree

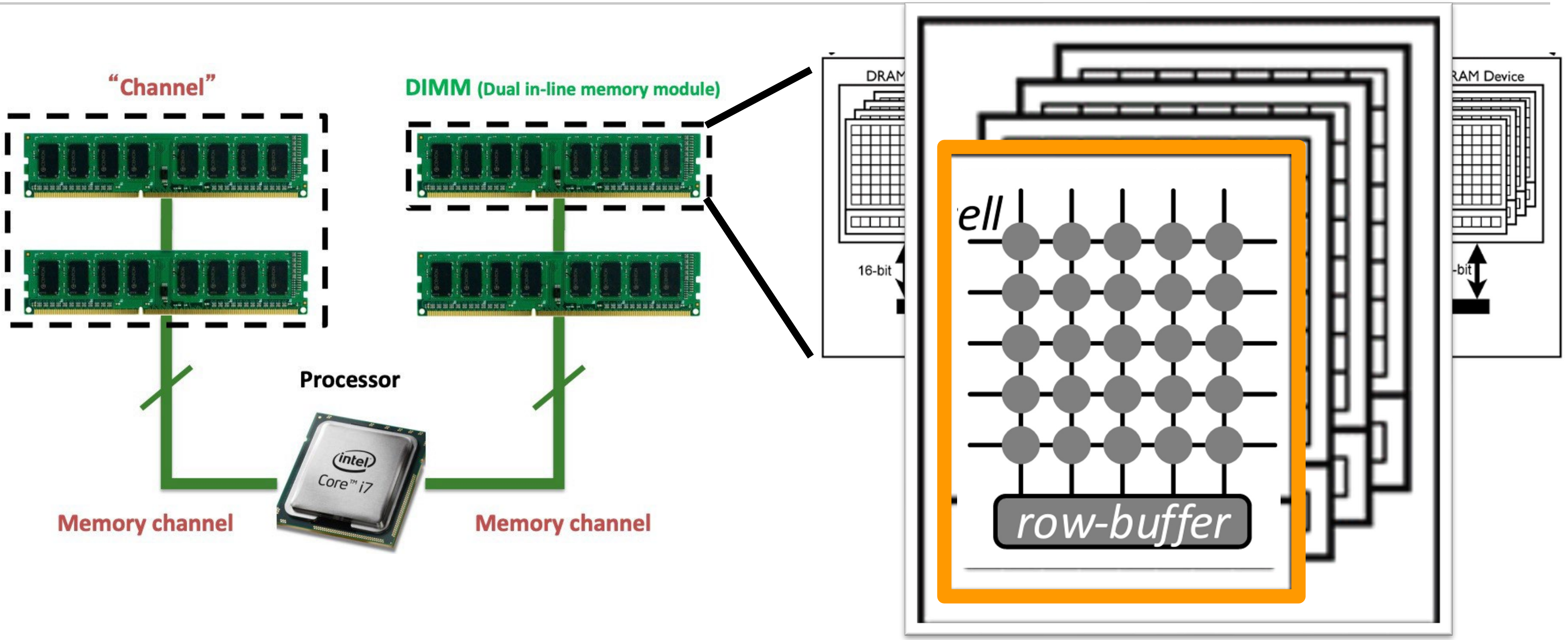
- Only need to store the root node on chip
- How do we verify block B1?
- Write to block B3?


Secure processor (trusted)
root
$\text{root} = \text{Hash}(f_{2i} \,||\, f_{2i+1})$
$f_i = \text{Hash}(g_{2i} \,||\, g_{2i+1})$
$g_i = \text{Hash}(h_{2i} \,||\, h_{2i+1})$
$h_i = \text{Hash}(B_i)$
f_0
f_1
g_0
g_1
g_2
g_3
h_0
h_1
h_2
h_3
h_4
h_5
h_6
h_7
B_0
B_1
B_2
B_3
B_4
B_5
B_6
B_7

Summary

- What can privileged software attackers do?
- Design tradeoffs between TCB size, flexibility, perf overhead, cost, etc.
  - Intel SGX, AMD SEV, ARM CCA
  - Keystone, Sanctum, Penglai, etc

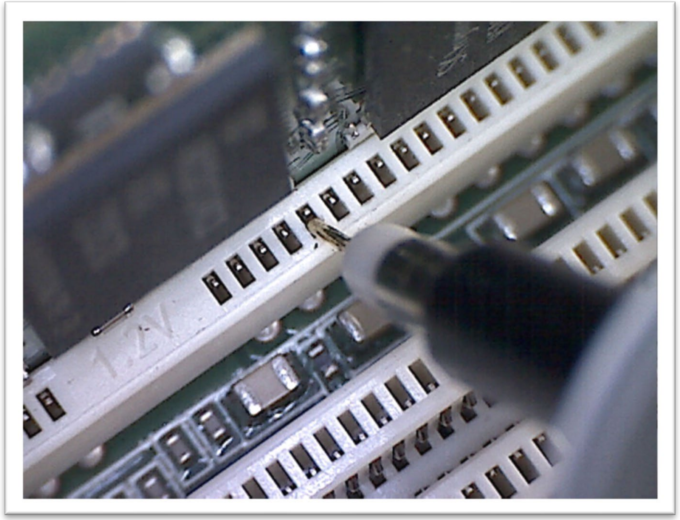
DRAM Organization: Top-down View

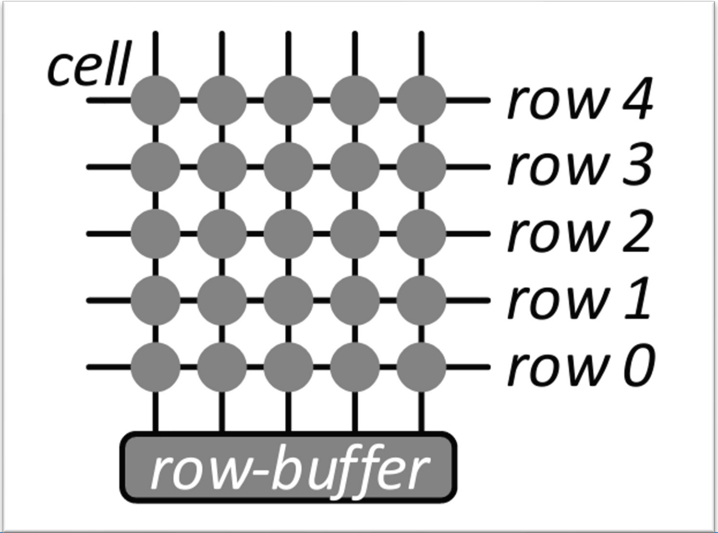


Channel -> DIMM -> Rank -> Bank -> Row/Column

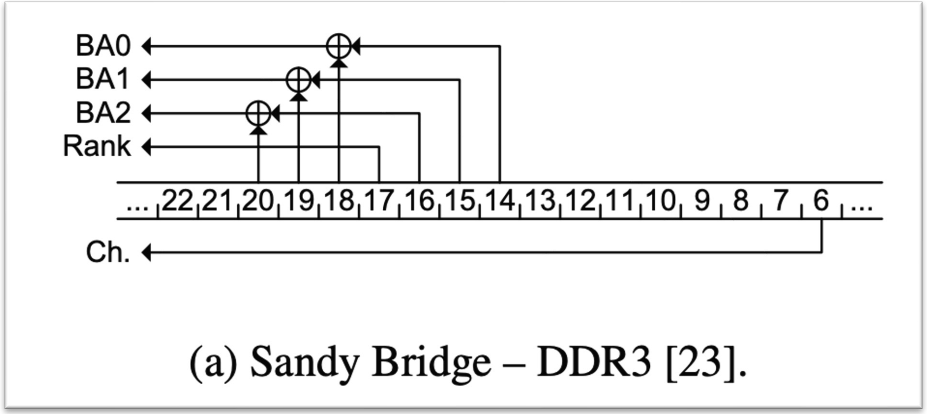
Reverse Engineer the Mapping

- Approach #1: Physical Probe
- Approach #2: Timing Side Channel via Row Buffer

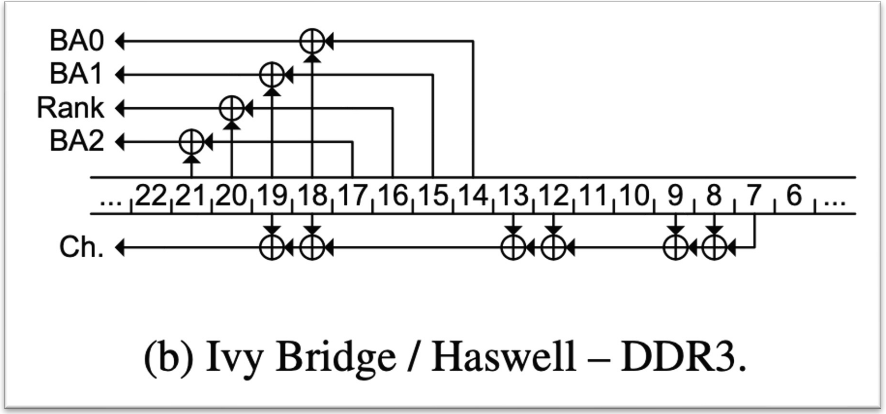




Address Mapping Examples



(a) Sandy Bridge – DDR3 [23].



(b) Ivy Bridge / Haswell – DDR3.

Pessl et al. DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks. USENIX'16

Rowhammer Attacks

Native Client (NaCl) Sandbox Escape

- NaCl is a sandbox for running native code (C/C++)
- Runs a “safe” subset of x86, statically verifying an executable
- Use bit flips to make an instruction sequence unsafe!

Example “Safe” Code:

```
andl $~31, %eax  // Truncate address to 32 bits
                 // and mask to be 32-byte-aligned.
addq %r15, %rax  // Add %r15, the sandbox base address.
jmp *%rax        // Indirect jump.
```

Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn and Dullien)

Native Client (NaCl) Sandbox Escape

We can flip bits to allow for (unsafe) non 32-byte-aligned jumps!

Exploited "Safe" Code:

```
andl $~31, %ecx   // Truncate address to 32 bits
                  // and mask to be 32-byte-aligned.
addq %r15, %rax   // Add %r15, the sandbox base address.
jmp *%rax         // Indirect jump.
```

Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn and Dullien)

Kernel Privilege Escalation

What could happen if a user could gain direct write access to a page table?

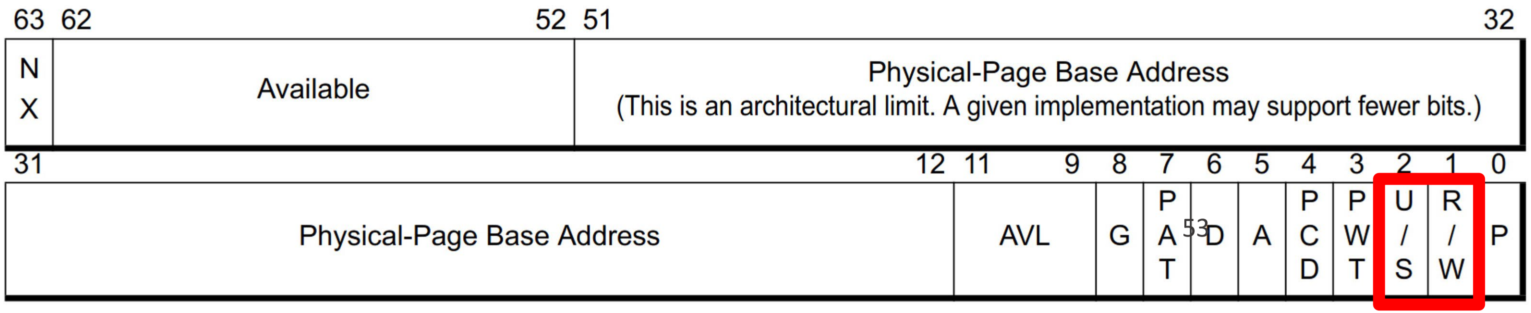

63 62 52 51 32
NX
Available
Physical-Page Base Address
(This is an architectural limit. A given implementation may support fewer bits.)
31 12 11 9 8 7 6 5 4 3 2 1 0
Physical-Page Base Address
AVL
G
PAT
D
A
PCD
PWT
U/S
R/W
P

Figure 5-21. 4-Kbyte PTE—Long Mode

Other Attacks

- Virtual machine takeover
  - Use page de-duplication to corrupt host machine
- OpenSSH attacks
  - Overwrite internal public key with attacker controlled one
  - Read private key directly (RAMBleed)
- Drammer
  - Rowhammer privilege escalation on ARM
  - Utilizes determinism in page allocation to target vulnerable DRAM rows
- Rowhammer.js
  - Remote takeover of a server vulnerable to rowhammer

Without memory integrity, *any* software-based security mechanism is insecure!

Rowhammer Mitigations?

- Manufacturing “better” chips

cost

- Increasing refresh rate

Performance, power

- Error Correcting Codes

cost, power

- Targeted row refresh (TRR) - Used in DDR4!

cost, power, complexity

- Retiring vulnerable cells

cost, power, complexity

- Static binary analysis

security

- User/kernel space isolation in physical memory

Error Correcting Codes (ECC)

- **Basic Idea:** Store extra *redundant* bits to be used in case of a flip!
- **Naive Implementation:** Store multiple copies and compare
- **Actual Implementation:** Hamming codes

Hamming codes allow for *single-error* correction, double error detection (aka **SECDED**)

How about more than 2-bit flips?

THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL