SGX Analysis

Chenglu Jin, S. K. Haider, M. Ahmad and H. Omar
Department of Electrical & Computer Engineering
University of Connecticut
Email: chenglu.jin@uconn.edu

Outline

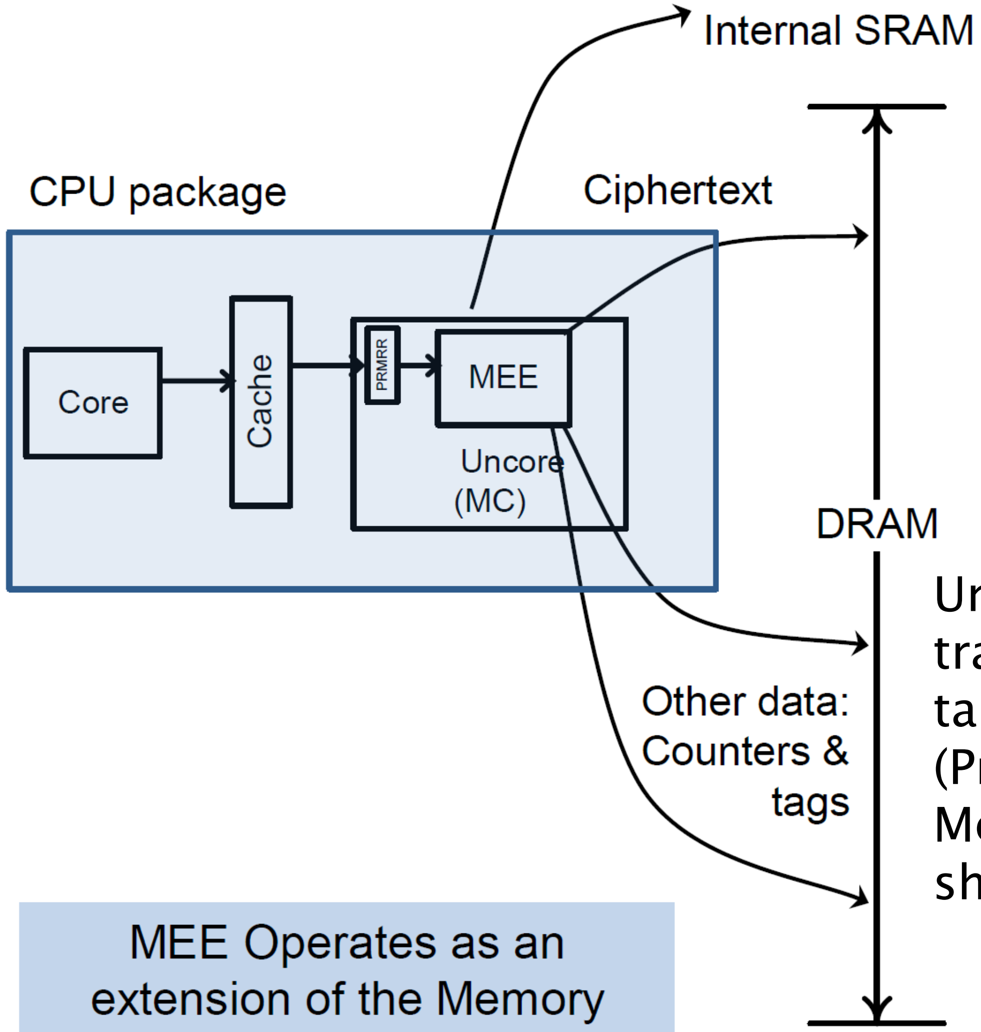
- SGX Memory Encryption Engine (MEE)
- SGX Memory Access Protection
- Tracking TLB Flushes
- Enclave Signature Verification
- SGX Security Properties
- Misconceptions about SGX
- Interaction with Anti-Virus Software

Memory Encryption Engine

- Memory Encryption Engine (MEE):
  - Added in the uncore part of the processor (Memory Controller)
  - protects SGX's Enclave Page Cache against physical attacks.
    - Data Confidentiality: Collections of memory images of 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.

How the MEE works – in a nutshell

- Core issues a transaction
  - (to MEE region); e.g., WRITE
- Transaction misses caches and forwarded to **M**emory **C**ontroller
- MC detects address belongs to MEE region & routes transaction to MEE
- Crypto processing and… …
- MEE **initiates additional memory accesses** to obtain (or write to) necessary data from DRAM
  - Produces plaintext (ciphertext)
  - Computes authentication tags
  - (uses/updates internal data)
  - writes ciphertext + added data



MEE Operates as an extension of the Memory Controller (MC)

Unauthorized DMA transactions that target the PRM (Processor Reversed Memory) range should be aborted.

MEE basic setup and policy

- Memory access always at 512 bits **C**ache **L**ine (**CL**) granularity
- Keys: randomly generated at reset by a HW DRNG module
  - Accessible only to MEE hardware

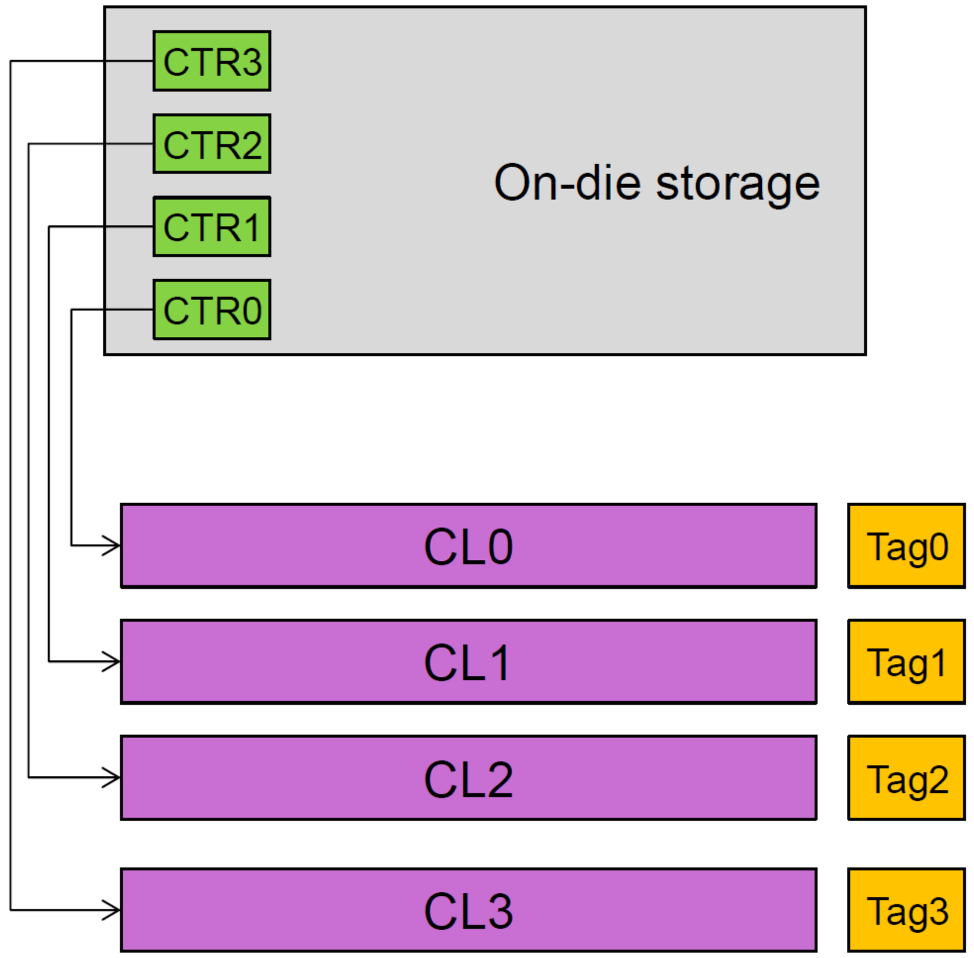
- Drop-and-lock policy: upon MAC tag mismatch, MEE
    - **Drops** the transaction (i.e., **no data is sent to the LLC**)
    - **Locks** the MC (i.e., **no further transactions are serviced**).
    - Eventually system **halts & reset is required** (with new keys)

Encryption Key: 128 bits
MAC Key: 128 bits
Hash Key: 512 bits

Message Authentication Code

- MAC can be used to protected memory integrity.
- But what is the problem if we only use MAC?
- Reply attack
- Solutions:
- 1. Hash Tree (Store updated root hash in TCB)
  - One root hash for the whole memory
- 2. Stateful MAC (Store updated states in TCB)
  - One state for each cache line
  - How to store all the states efficiently???

One level data structure



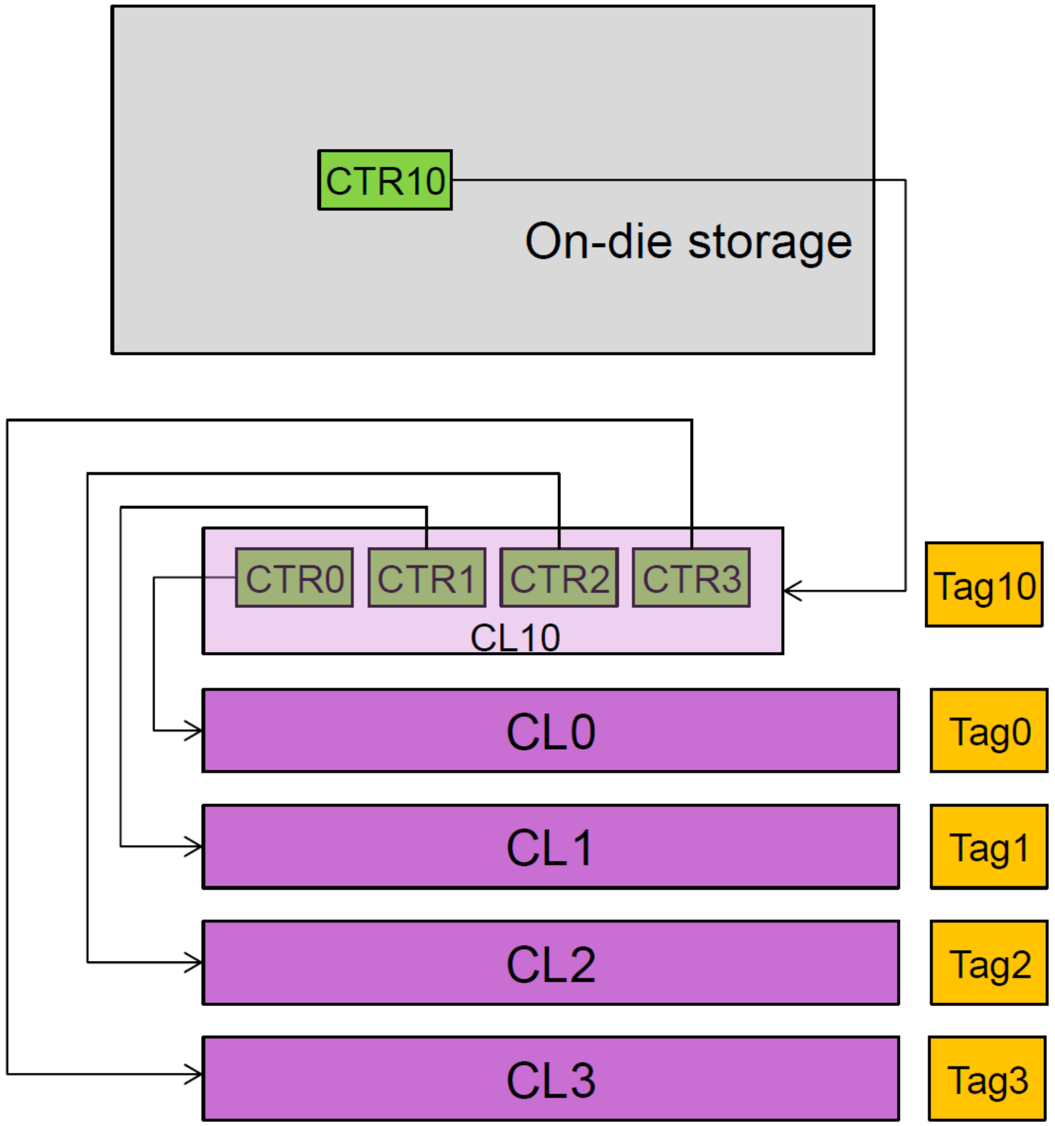
Tag = MAC (CTR, CL)

CTR is trusted

Integrity + freshness

Too many counters in trusted region. Too expensive!

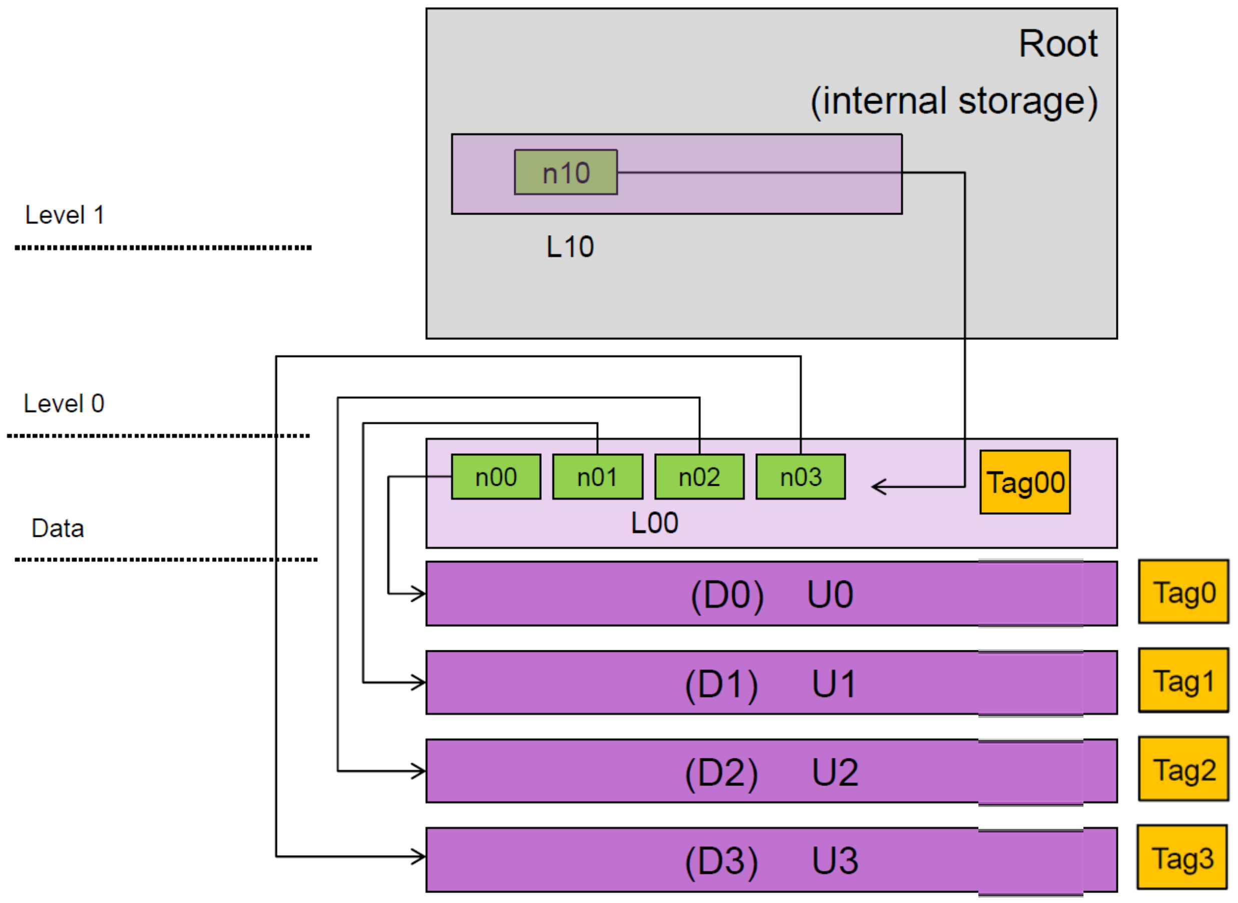
Compressing it: a 2-level data structure



- "**Stateful**" MAC over Data and CTR
- 1st level tags protect Data
- 2nd level tag protects the counters
- Top level tag is internal → trusted
- Counters protect "freshness"

- **Trading internal storage with a walk over the data structure**
- **(complexity & performance)**

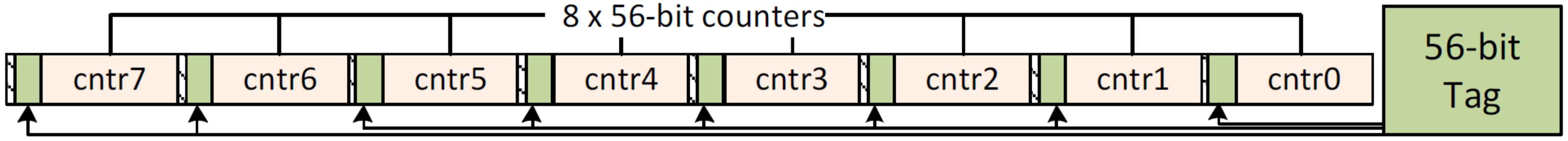
Embedded MAC tags



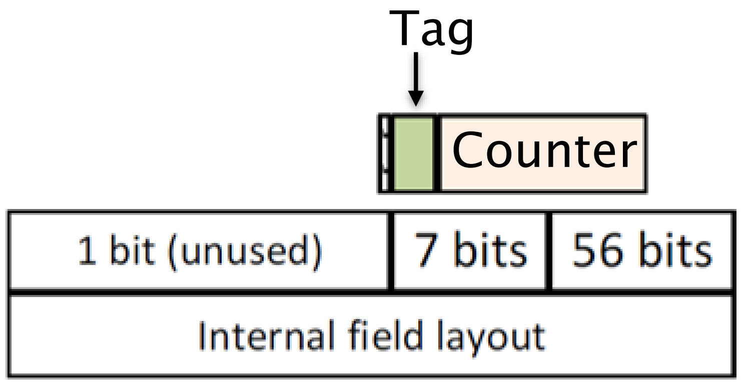
Embedded MAC tags into counter cache line to save the memory accesses.

Why don't we embed tags into data cache lines as well?

A Counter Cache Line

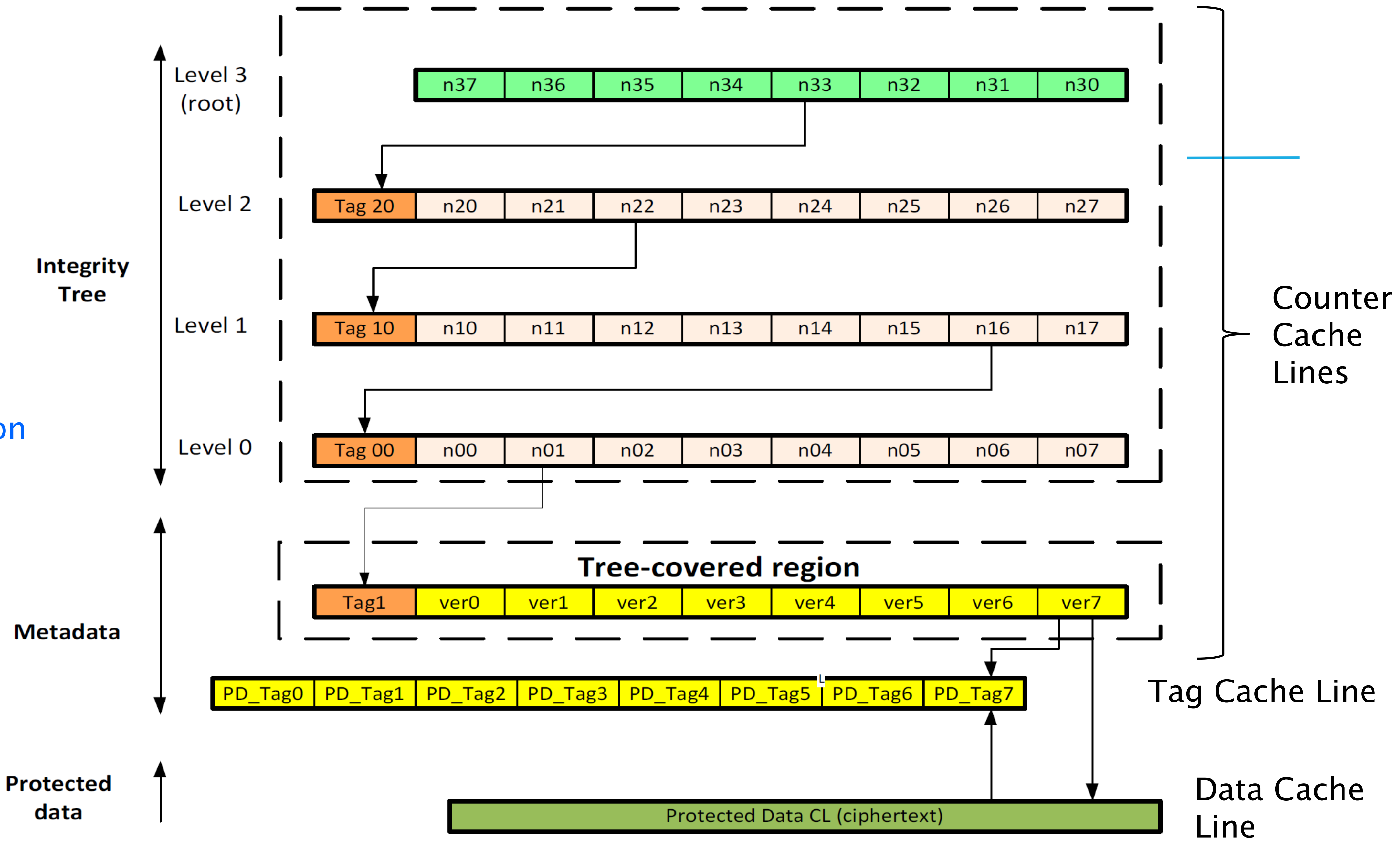


56-bit counters
56-bit tags



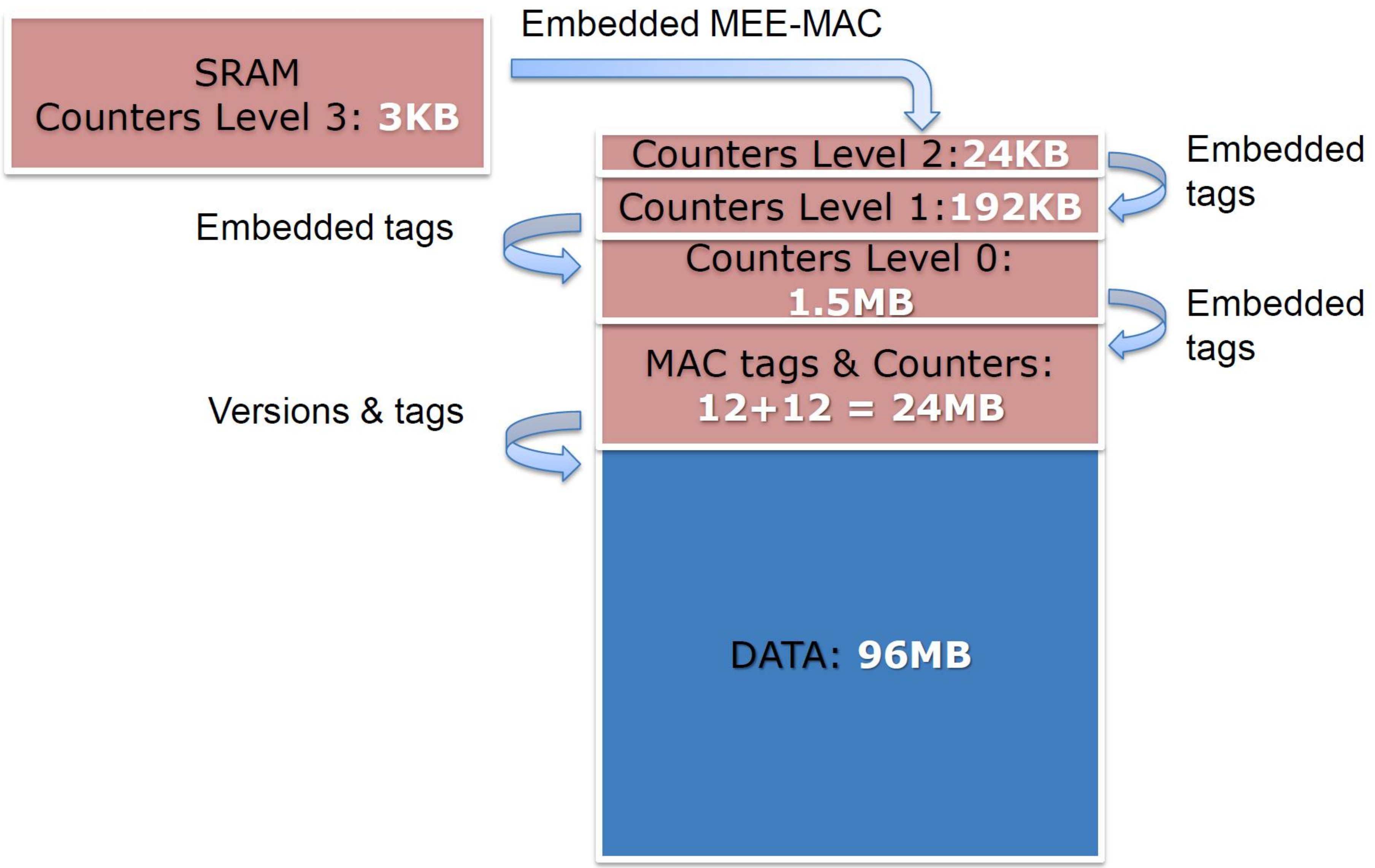
One CL accommodates 8 counters and **embedded tag**

$56 * 8 + 56 + 8 = 512$

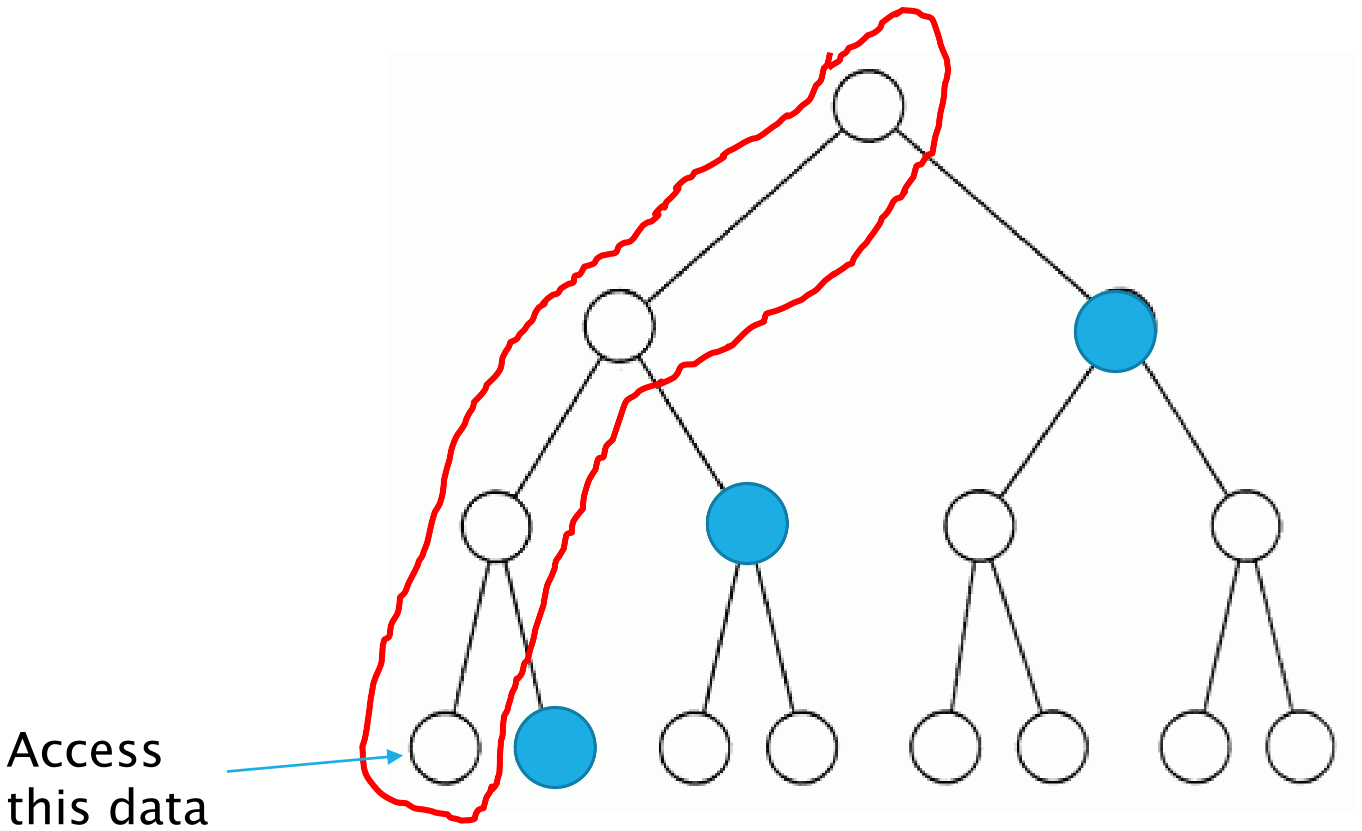


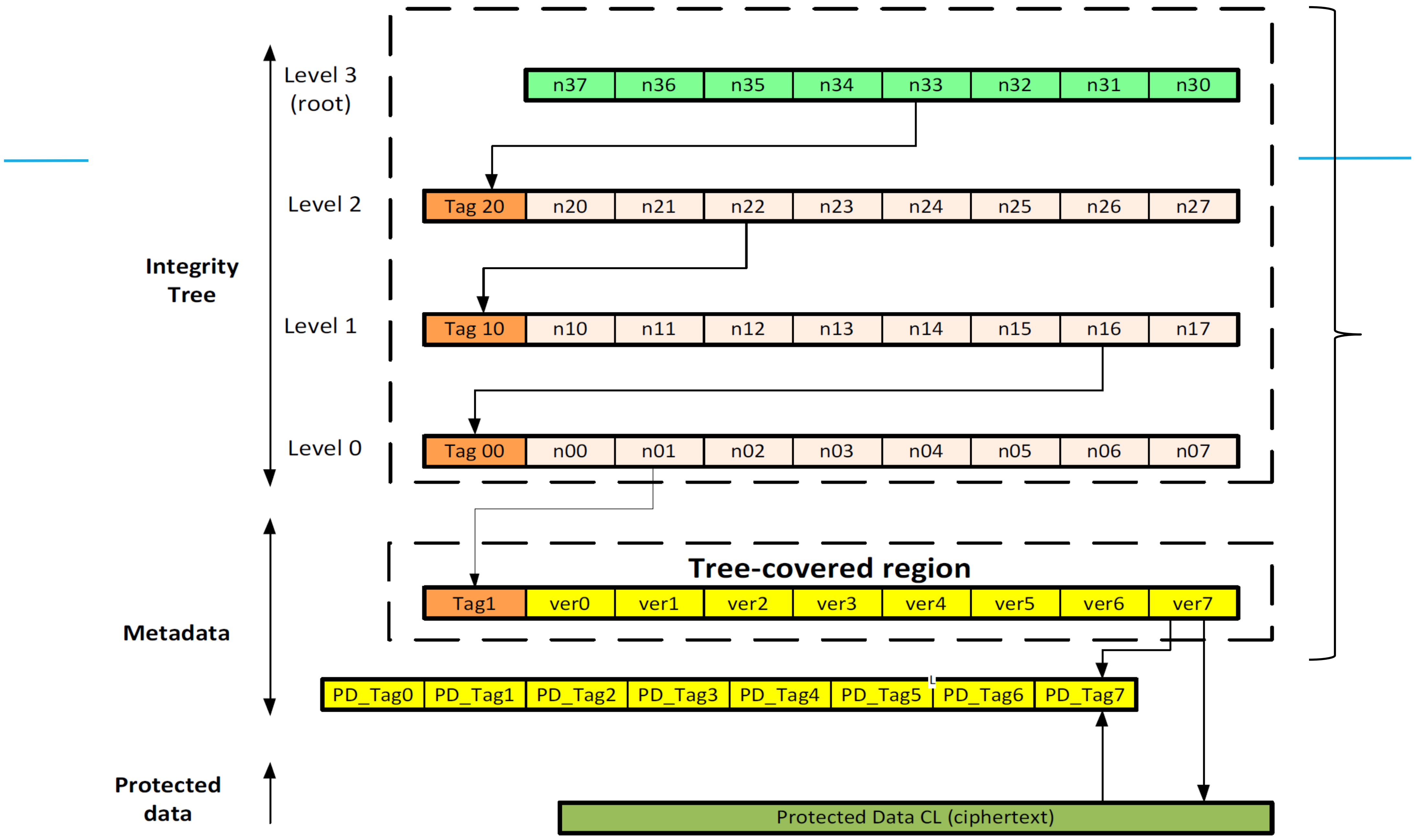
What is the compression rate?

The overall compression rate



Comparison with Hash Tree





Integrity Tree
Level 3 (root)
n37
n36
n35
n34
n33
n32
n31
n30
Level 2
Tag 20
n20
n21
n22
n23
n24
n25
n26
n27
Level 1
Tag 10
n10
n11
n12
n13
n14
n15
n16
n17
Level 0
Tag 00
n00
n01
n02
n03
n04
n05
n06
n07
Metadata
Tree-covered region
Tag1
ver0
ver1
ver2
ver3
ver4
ver5
ver6
ver7
PD_Tag0
PD_Tag1
PD_Tag2
PD_Tag3
PD_Tag4
PD_Tag5
PD_Tag6
PD_Tag7
Protected data
Protected Data CL (ciphertext)

More details

- How to encrypt?

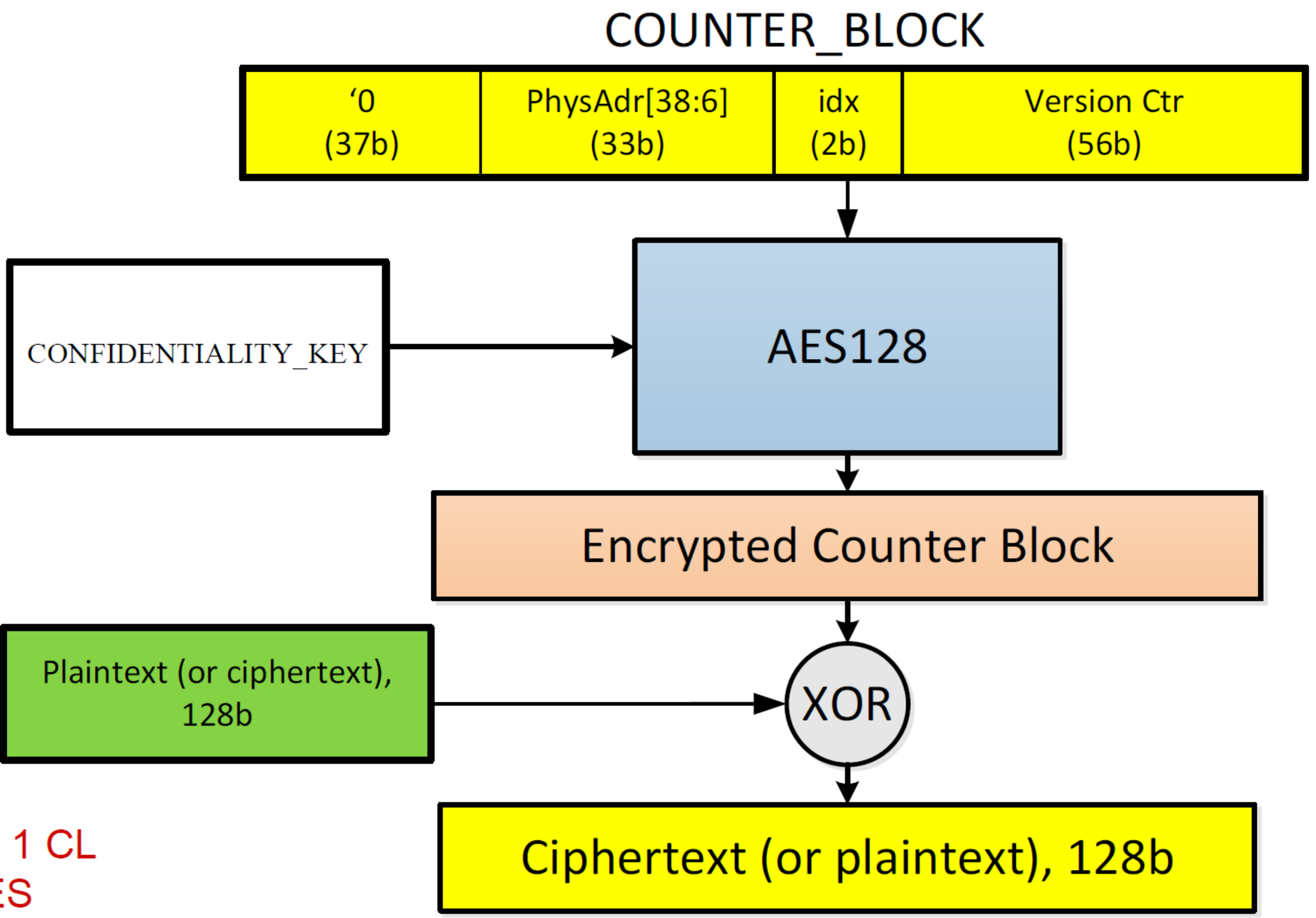
- How to compute MAC?

- Background: AES (Advanced Encryption Standard)
- AES-128: 128-bit plaintext, 128-bit ciphertext, 128-bit key

MEE Counter Mode

Spatial and temporal coordinates identify every 16B block in the address space, at any time

Address has 39 bits; idx: 2 bits representing location in the CL; Version: 56 bits



Encryption of 1 CL involves 4 AES operations

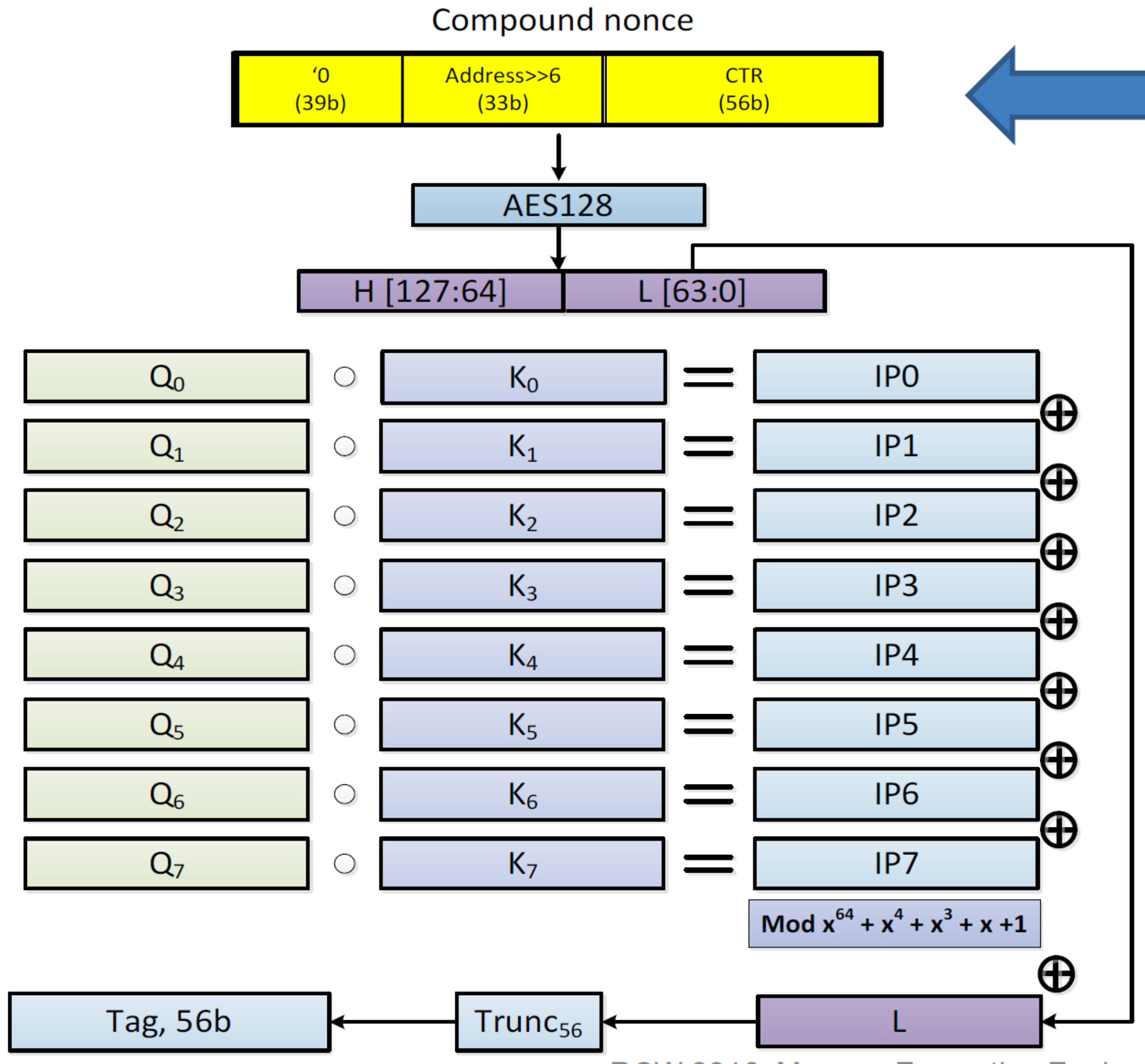
Confidentiality Bound

Proposition 1 (Confidentiality bound). *Let* **Adv** *be the advantage of a probabilistic polynomial time algorithm in distinguishing the ciphertexts in T' from a set of random strings. Then,*

$$\mathbf{Adv} \leq \varepsilon_{AES}(q') + \frac{(q')^2}{2^{125}} \tag{4}$$

The MAC algorithm

Tag = L + $Q_0 \bullet K_0 + Q_1 \bullet K_1 + Q_2 \bullet K_2 + \ldots + Q_7 \bullet K_7$ in GF(2^{64}) Truncated to 56 bits



Spatial & temporal coordinates

- Multilinear universal hash
  - ("Inner Product hash")
  - Operations in GF (2^{64})
- Masked by (truncated) AES
- Truncated to 56 bits
  - Why? Real world…
  - If tags and counters have same length they can share same internal bus

17

MEE Forgery Resistance

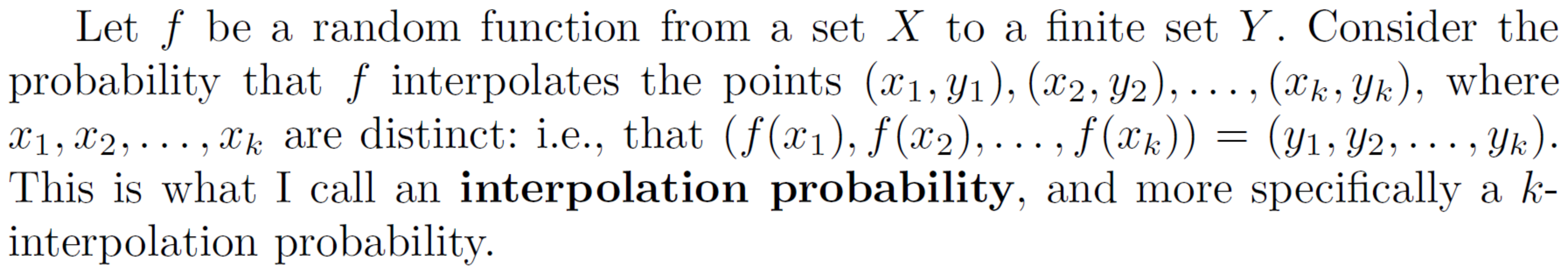
Proposition 2 (The MEE forgery resistance). *An active adversary who collects a trace of $q \leq 2^{56} - 2$ message-tag samples that the MEE produces, and attempts a forgery, has success probability at most*

$$P_{success}(q) = \varepsilon_{AES}(q) + \varepsilon \cdot \left(1 + \frac{q^2}{2^{128}}\right) \leq$$

$$\leq \varepsilon_{AES}(2^{56}) + \frac{1}{2^{56}} \cdot \left(1 + \frac{1}{2^{16}}\right) \qquad (11)$$

Proof

- Use the main theorem in [5], which proves the security bounds for such a MAC construction for us.
- First, we need to compute Maximum Interpolation Probability for function f(b) = Truncate$_t$(AES(K,b))



Let f be a random function from a set X to a finite set Y. Consider the probability that f interpolates the points $(x_1, y_1), (x_2, y_2), \ldots, (x_k, y_k)$, where $x_1, x_2, \ldots, x_k$ are distinct: i.e., that $(f(x_1), f(x_2), \ldots, f(x_k)) = (y_1, y_2, \ldots, y_k)$. This is what I call an **interpolation probability**, and more specifically a k-interpolation probability.

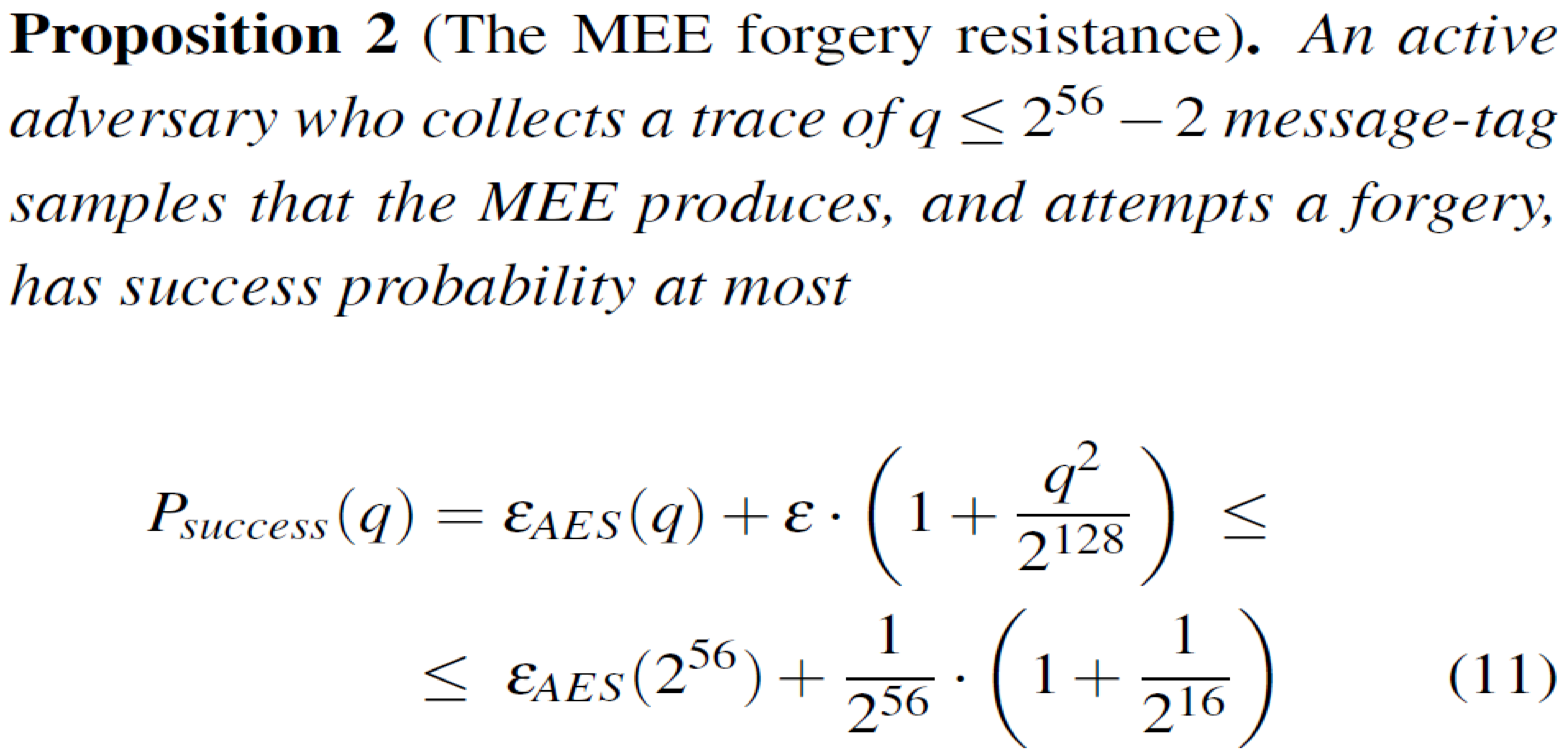
[5] Bernstein D J. Stronger security bounds for Wegman-Carter-Shoup authenticators[M] EUROCRYPT'05

Inequalities needed in proof

- (1)
- $(1 - a_1) * (1 - a_2) * \cdots * (1 - a_k) \geq 1 - (a_1 + a_2 + \cdots + a_k)$ for any $a_j \geq 0$ such that $\sum_{j=1}^{k} a_j \leq 1$

- (2)
- $\frac{1}{1-w} \leq 1 + 2w$ for $0 < w \leq 0.5$

MEE Forgery Resistance



Proposition 2 (The MEE forgery resistance). *An active adversary who collects a trace of $q \leq 2^{56} - 2$ message-tag samples that the MEE produces, and attempts a forgery, has success probability at most*

$$P_{success}(q) = \varepsilon_{AES}(q) + \varepsilon \cdot \left(1 + \frac{q^2}{2^{128}}\right) \leq$$
$$\leq \varepsilon_{AES}(2^{56}) + \frac{1}{2^{56}} \cdot \left(1 + \frac{1}{2^{16}}\right) \qquad (11)$$

In order to maximize $P_{success}$, the attacker need to collect 2^{56} – 2 MACs. But due to the cost of collecting the trace, an efficient strategy for an attacker would be blind guessing with probability $1/2^{56}$

Are 56-bit tags and 56-bit counters secure enough?

- Rollover 56-bit counter -> 10.5 years
- Forgery 56-bit tag -> 2M years
  - Assuming 1000 forge root per second.

Worried?

Outline

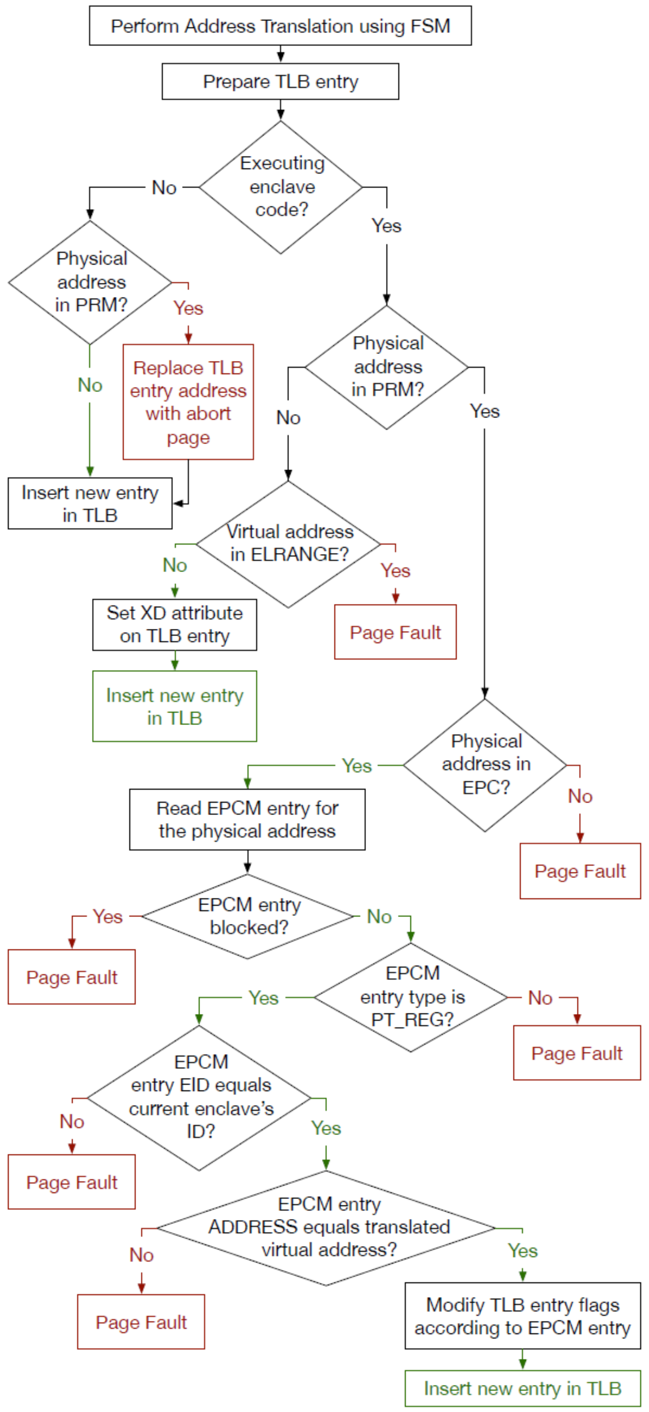
- SGX Memory Encryption Engine (MEE)
- SGX Memory Access Protection
- Tracking TLB Flushes
- Enclave Signature Verification
- SGX Security Properties
- Misconceptions about SGX
- Interaction with Anti-Virus Software

SGX Memory Access Protection

- MEE sits in MC, it cannot protect an enclave’s memory from software attacks.
- The root of SGX’s protections against software attacks is memory access checks which prevents the currently running software from accessing memory that does not belong to it.
- Implemented in Page Miss Handler (PMH)
  - PMH triggers the extra microcode for all address translations
  - All the SGX instructions are implemented in microcode, which introduces many new registers for storing metadata of enclave.

Security Check for Memory Access

SGX adds a few security checks to the PMH. The checks ensure that all the TLB entries created by the address translation unit meet SGX's memory access restrictions.



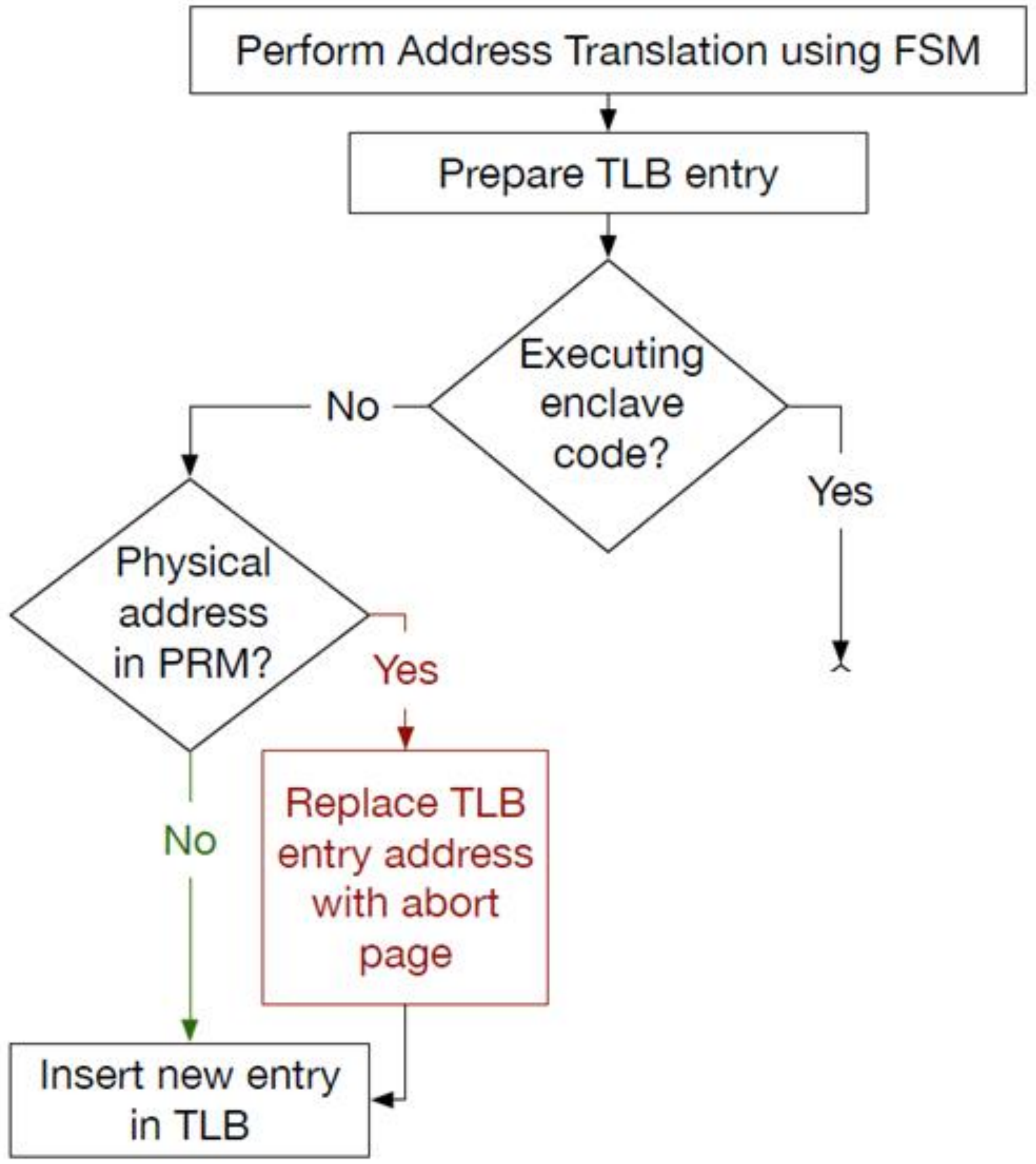
Perform Address Translation using FSM
Prepare TLB entry
Executing enclave code?
No
Yes
Physical address in PRM?
Yes
No
Replace TLB entry address with abort page
Insert new entry in TLB
Physical address in PRM?
No
Yes
Virtual address in ELRANGE?
No
Yes
Set XD attribute on TLB entry
Page Fault
Insert new entry in TLB
Physical address in EPC?
Yes
No
Read EPCM entry for the physical address
Page Fault
EPCM entry blocked?
Yes
No
Page Fault
EPCM entry type is PT_REG?
Yes
No
Page Fault
EPCM entry EID equals current enclave's ID?
No
Yes
Page Fault
EPCM entry ADDRESS equals translated virtual address?
No
Yes
Page Fault
Modify TLB entry flags according to EPCM entry
Insert new entry in TLB

SGX Security Check Correctness

- Top-level invariant: At all times, all the TLB entries in every logical processor will be consistent with SGX's security guarantees.
- First breakdown the top level invariant into three cases on:
  - whether a logical processor (LP) is executing enclave code or not
  - whether the TLB entries translate virtual addresses in the current enclave's ELRANGE

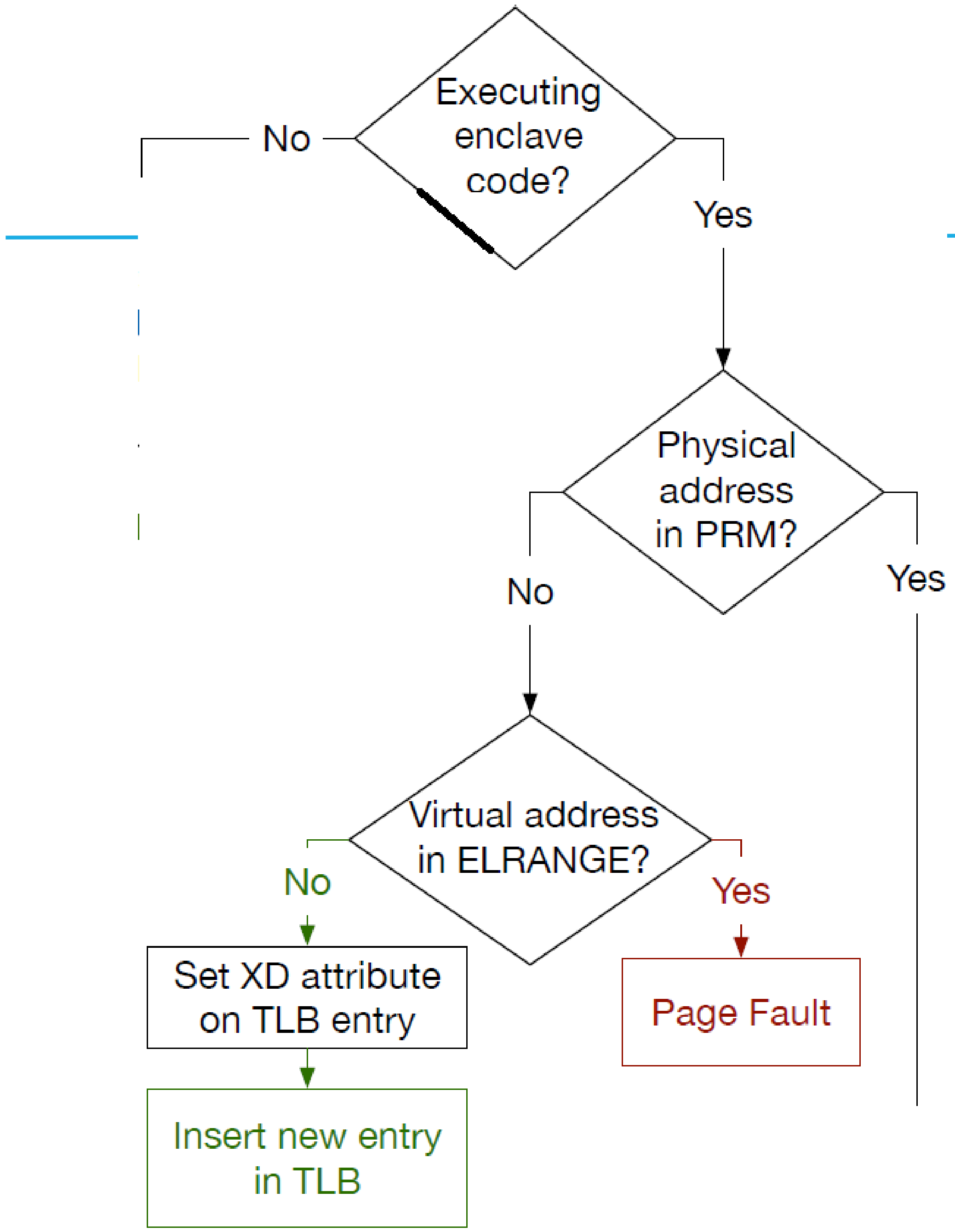
Case Invariants

- 1. At all times when an LP is outside enclave mode, its TLB may only contain physical addresses belonging to DRAM pages outside the PRM.
- 2. At all times when an LP is inside enclave mode, the TLB entries for virtual addresses outside the current enclave's ELRANGE must contain physical addresses belonging to DRAM pages outside the PRM.
- 3. At all times when an LP is in enclave mode, the TLB entries for virtual addresses inside the current enclave's ELRANGE (Enclave Linear Address Range) must match the virtual memory layout specified by the enclave author.



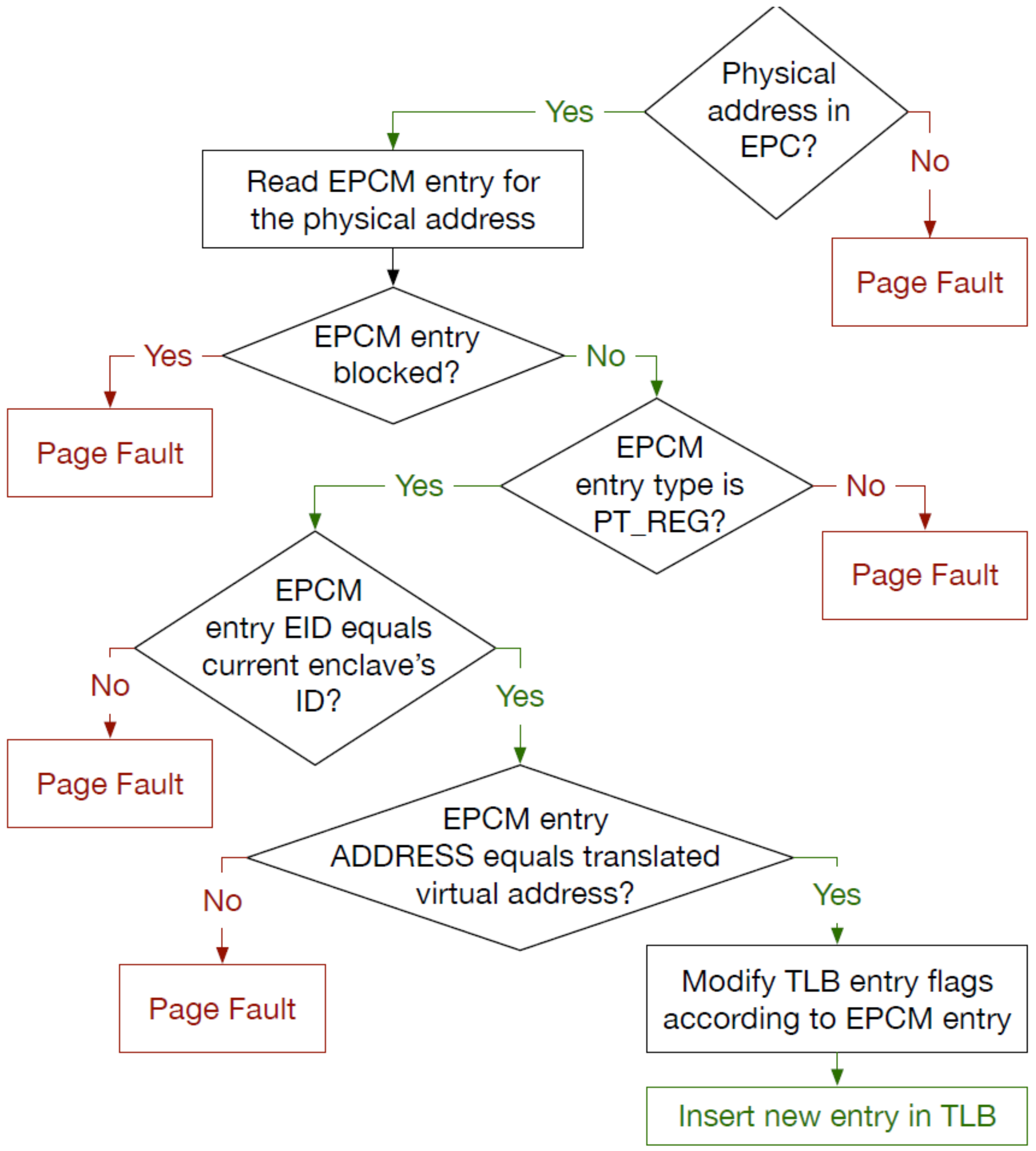
Invariant 1

- At all times when an LP is outside enclave mode, its TLB may only contain physical addresses belonging to DRAM pages outside the PRM.



Invariant 2

At all times when an LP is inside enclave mode, the TLB entries for virtual addresses outside the current enclave's ELRANGE (Enclave Linear Address Range) must contain physical addresses belonging to DRAM pages outside the PRM.


Physical address in EPC?
Yes
No
Read EPCM entry for the physical address
Page Fault
EPCM entry blocked?
Yes
No
Page Fault
EPCM entry type is PT_REG?
Yes
No
Page Fault
EPCM entry EID equals current enclave's ID?
No
Yes
Page Fault
EPCM entry ADDRESS equals translated virtual address?
No
Yes
Page Fault
Modify TLB entry flags according to EPCM entry
Insert new entry in TLB

Invariant 3

- At all times when an LP is in enclave mode, the TLB entries for virtual addresses inside the current enclave's ELRANGE (Enclave Linear Address Range) must match the virtual memory layout specified by the enclave author.

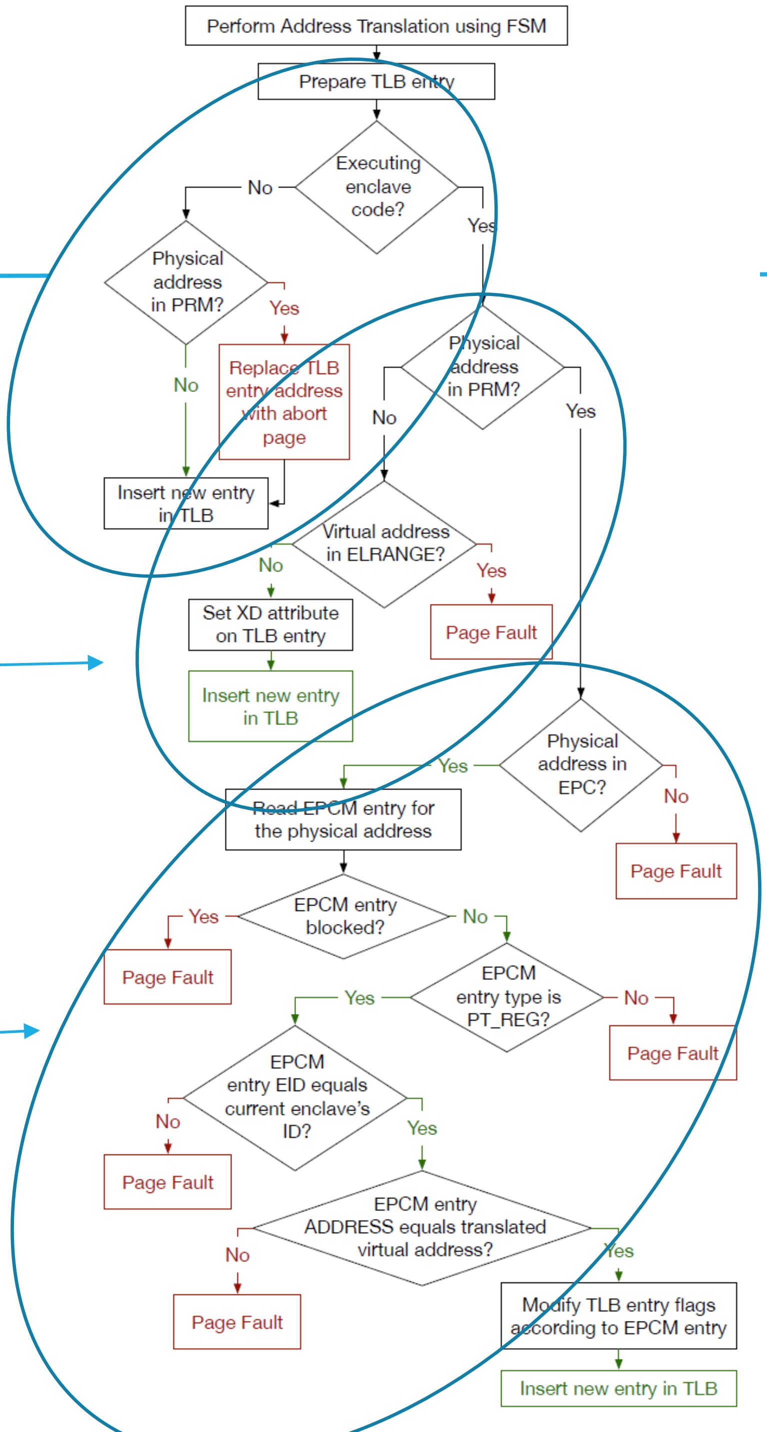
The entire flow

Top-level invariant: At all times, all the TLB entries in every logical processor will be consistent with SGX's security guarantees.

Invariant 1

Invariant 2

Invariant 3



Perform Address Translation using FSM
Prepare TLB entry
Executing enclave code?
No
Yes
Physical address in PRM?
Yes
No
Replace TLB entry address with abort page
Insert new entry in TLB
Physical address in PRM?
No
Yes
Virtual address in ELRANGE?
No
Yes
Set XD attribute on TLB entry
Page Fault
Insert new entry in TLB
Physical address in EPC?
Yes
No
Read EPCM entry for the physical address
Page Fault
EPCM entry blocked?
Yes
No
Page Fault
EPCM entry type is PT_REG?
Yes
No
Page Fault
EPCM entry EID equals current enclave's ID?
No
Yes
Page Fault
EPCM entry ADDRESS equals translated virtual address?
No
Yes
Page Fault
Modify TLB entry flags according to EPCM entry
Insert new entry in TLB

Outline

- SGX Memory Encryption Engine (MEE)
- SGX Memory Access Protection
- Tracking TLB Flushes
- Enclave Signature Verification
- SGX Security Properties
- Misconceptions about SGX
- Interaction with Anti-Virus Software

Tracking TLB Flushes

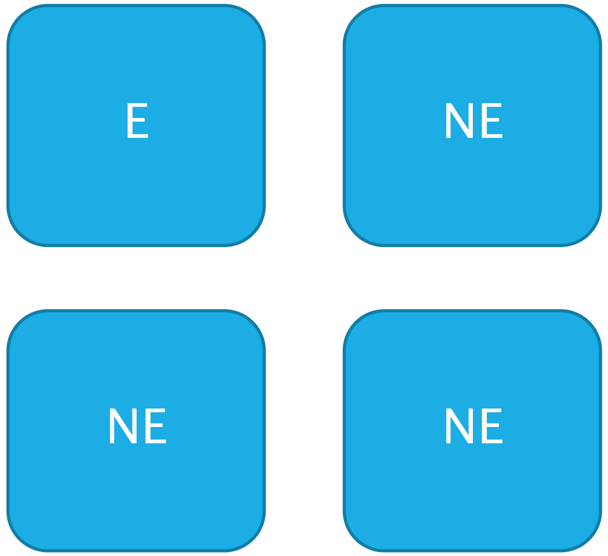
- Tracking TLB flushes is equivalent to verifying that all the logical processors have exited Enclave mode at least once after we start tracking.

- We rely on the SECS to store variables for tracking.

Tracking TLB Flushes

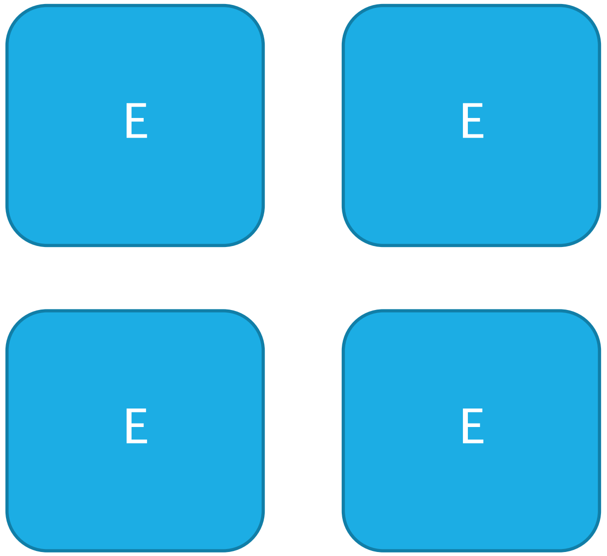
- ECREATE

- SECS.tracking = False
- SECS.done-tracking = False
- SECS.active-threads = 1
- SECS.tracked-threads = 0
- SECS.lp-mask = [0,0,0,0]



Tracking TLB Flushes

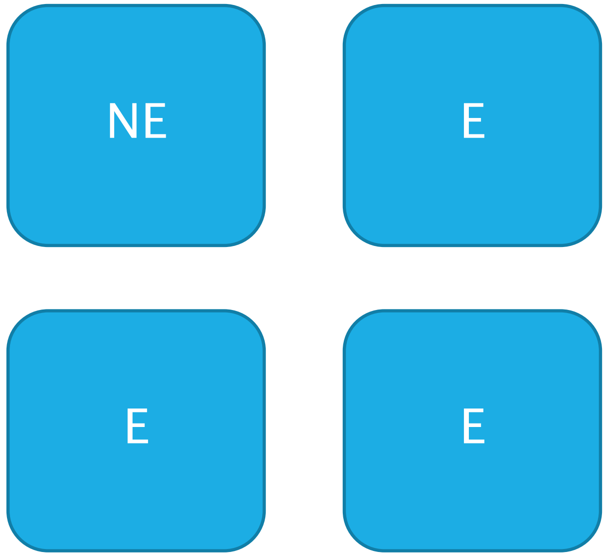
- ETRACK
  - Start of a TLB tracking cycle
- SECS.tracking = True
- SECS.done-tracking = False
- SECS.active-threads = 4
- SECS.tracked-threads = 4
- SECS.lp-mask = [0,0,0,0]



Tracking TLB Flushes

- EEXIT

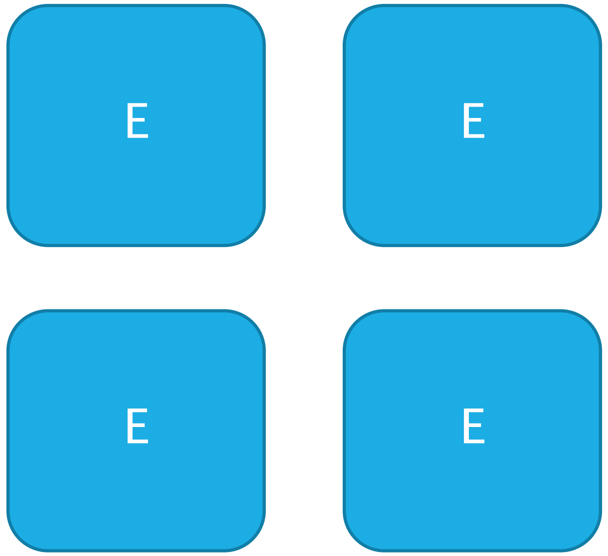
- SECS.tracking = True
- SECS.done-tracking = False
- SECS.active-threads = 3
- SECS.tracked-threads = 3
- SECS.lp-mask = [1,0,0,0]



Tracking TLB Flushes

- EENTER

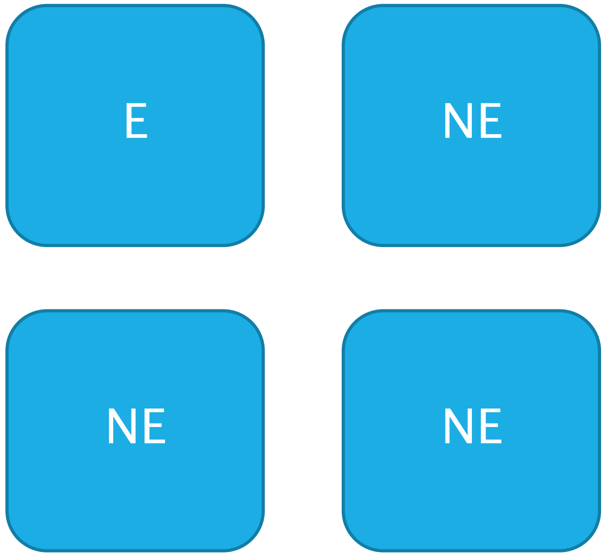
- SECS.tracking = True
- SECS.done-tracking = False
- SECS.active-threads = 4
- SECS.tracked-threads = 3
- SECS.lp-mask = [1,0,0,0]



Tracking TLB Flushes

- EEXIT

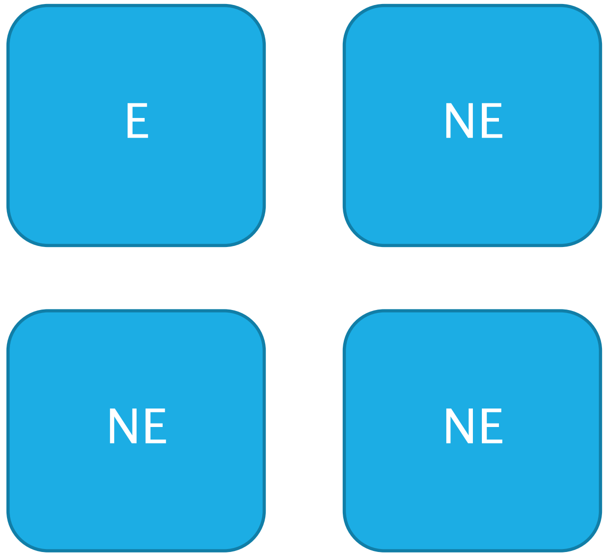
- SECS.tracking = True
- SECS.done-tracking = True
- SECS.active-threads = 1
- SECS.tracked-threads = 0
- SECS.lp-mask = [1,1,1,1]



Tracking TLB Flushes

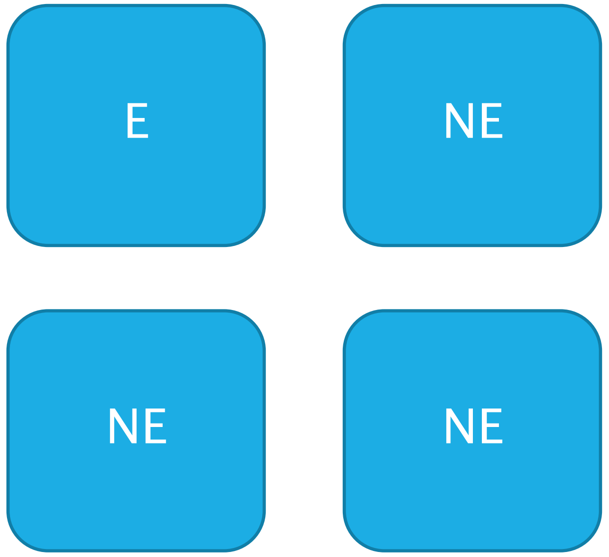
- EWB-VERIFY

- SECS.tracking = True
- SECS.done-tracking = True
- SECS.active-threads = 1
- SECS.tracked-threads = 0
- SECS.lp-mask = [1,1,1,1]



Tracking TLB Flushes

- EBLOCK
  - End of a TLB tracking cycle
- SECS.tracking = False
- SECS.done-tracking = True
- SECS.active-threads = 1
- SECS.tracked-threads = 0
- SECS.lp-mask = [1,1,1,1]



Outline

- SGX Memory Encryption Engine (MEE)
- SGX Memory Access Protection
- Tracking TLB Flushes
- Enclave Signature Verification
- SGX Security Properties
- Misconceptions about SGX
- Interaction with Anti-Virus Software

Enclave Signature Verification

- Let *m* be the public modulus in the enclave author's RSA key, and *s* be the enclave signature. Public exponent e is 3,
- Verifying the RSA signature $M = s^3 \bmod m$

RSA signature verification Algorithm

$$q_1 = \left\lfloor \frac{s^2}{m} \right\rfloor$$

$$q_2 = \left\lfloor \frac{s^3 - q_1 \times s \times m}{m} \right\rfloor$$

Avoid division and modulo operations.

1. Compute $u \leftarrow s \times s$ and $v \leftarrow q_1 \times m$
2. If $u < v$, abort. q_1 must be incorrect.
3. Compute $w \leftarrow u - v$
4. If $w \geq m$, abort. q_1 must be incorrect.
5. Compute $x \leftarrow w \times s$ and $y \leftarrow q_2 \times m$
6. If $x < y$, abort. q_2 must be incorrect.
7. Compute $z \leftarrow x - y$.
8. If $z \geq m$, abort. q_2 must be incorrect.
9. Output z.

$$0 \leq s^2 - q_1 \times m < m$$

$$0 \leq w \times s - q_2 \times m < m$$

$$\begin{aligned} z &= w \times s \bmod m \\ &= (s^2 \bmod m) \times s \bmod m \\ &= s^2 \times s \bmod m \\ &= s^3 \bmod m \end{aligned}$$

Outline

- SGX Memory Encryption Engine (MEE)
- SGX Memory Access Protection
- Tracking TLB Flushes
- Enclave Signature Verification
- SGX Security Properties
- Misconceptions about SGX
- Interaction with Anti-Virus Software

SGX Security Properties

- An isolated container whose contents receive special hardware protection that intended to translate into privacy, integrity and freshness guarantees.
- Offers a certificate-based identity system that can be used to migrate secrets between enclaves that have certificates issued by the same authority.

Physical Attacks

- Lack of publicly available details about the hardware implementation of SGX => some avenues for future exploration
- Port attack, especially Generic Debug eXternal Connection.
- Bus attack, because the data in cache is in plaintext.
- Bus tapping attack, because SGX does not hide the memory access patterns.
- Cache timing attack.
- Intel Management Engine may be not protected.
- Fused seal key. -> PUF
- Power analysis

Privileged Software Attacks

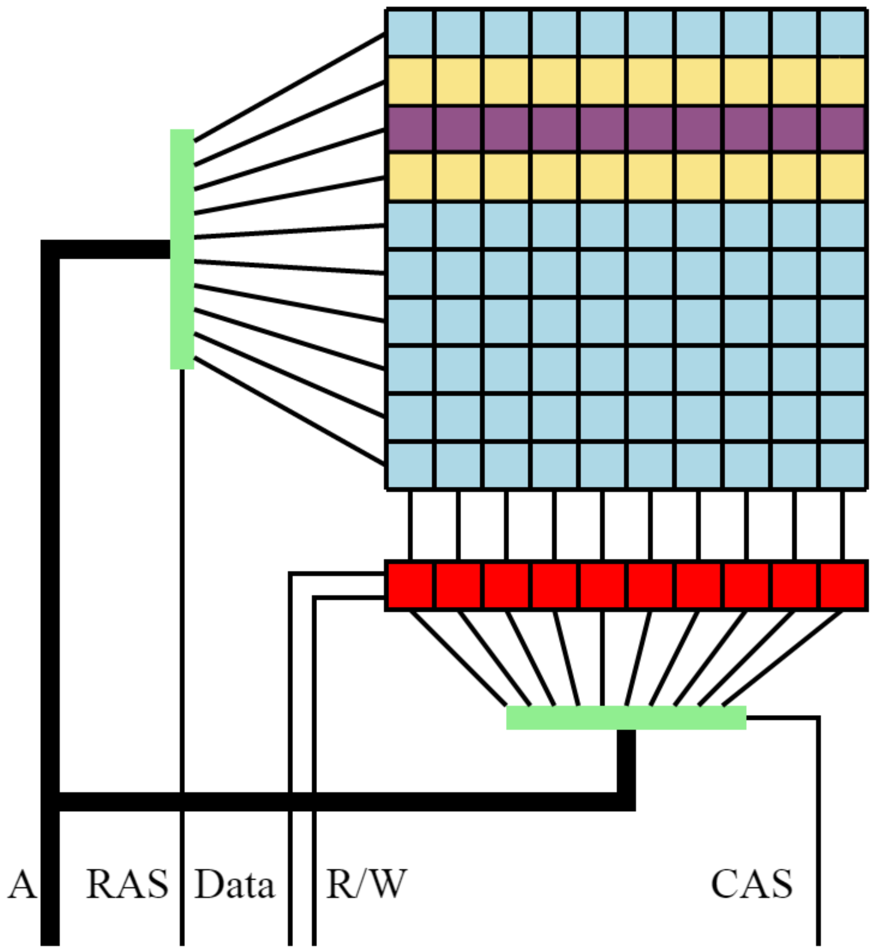
- The SGX design prevents malicious software from directly reading or from modifying the EPC pages that store an enclave's code and data.
- This relies on two pillars (isolation principle):
- First, the SGX implementation runs in the processor's microcode, which is effectively a higher privilege level that system software does not have access to.
- Second, SGX's microcode is always involved when a CPU transitions between enclave code and non-enclave code, and therefore regulates all interactions between system software and an enclave's environment

Memory Mapping Attacks

- SGX can prevent active attacks by rejecting undesirable address translations before they reach the TLB. Also, it prevents the active attacks using page swapping or stale TLB entries.
- Passive address translation attacks can learn the memory access patterns.

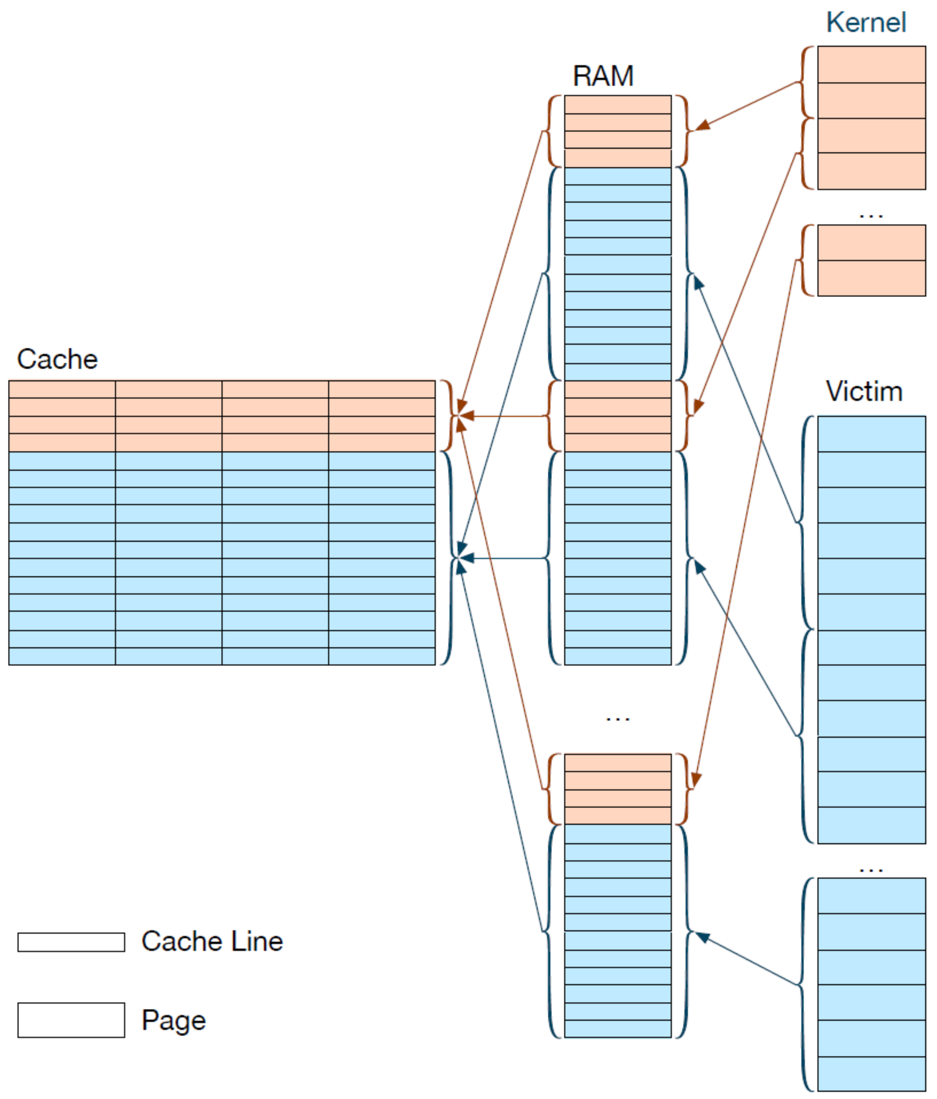
Software Attacks on Peripherals

- PCI (Peripheral Controller Interface) Express attacks are prevented, because MC rejects any DMA transfer that falls within the Processor Reserved Memory
- DRAM attacks (e.g. Rowhammer) are prevented due to MEE.
- Firmware attacks (especially, ME's firmware) are not mentioned in the documents. (ME compromise = DRAM attacks)
- SGX does not protect against software side-channel attacks that rely on performance counters (e.g. cache misses, branch predictors).



Cache Timing Attacks

- Cache timing attacks are not mentioned in the threat model.
- A malicious system software can make it worse.
  - Control the enclave scheduling
  - Control address translation
- SGX does not prevent this attack, but increases the difficulties: SGX's enclave entry implementation could flush the core's private caches.
- The Last Level Cache is still vulnerable, because it is shared among all the cores.

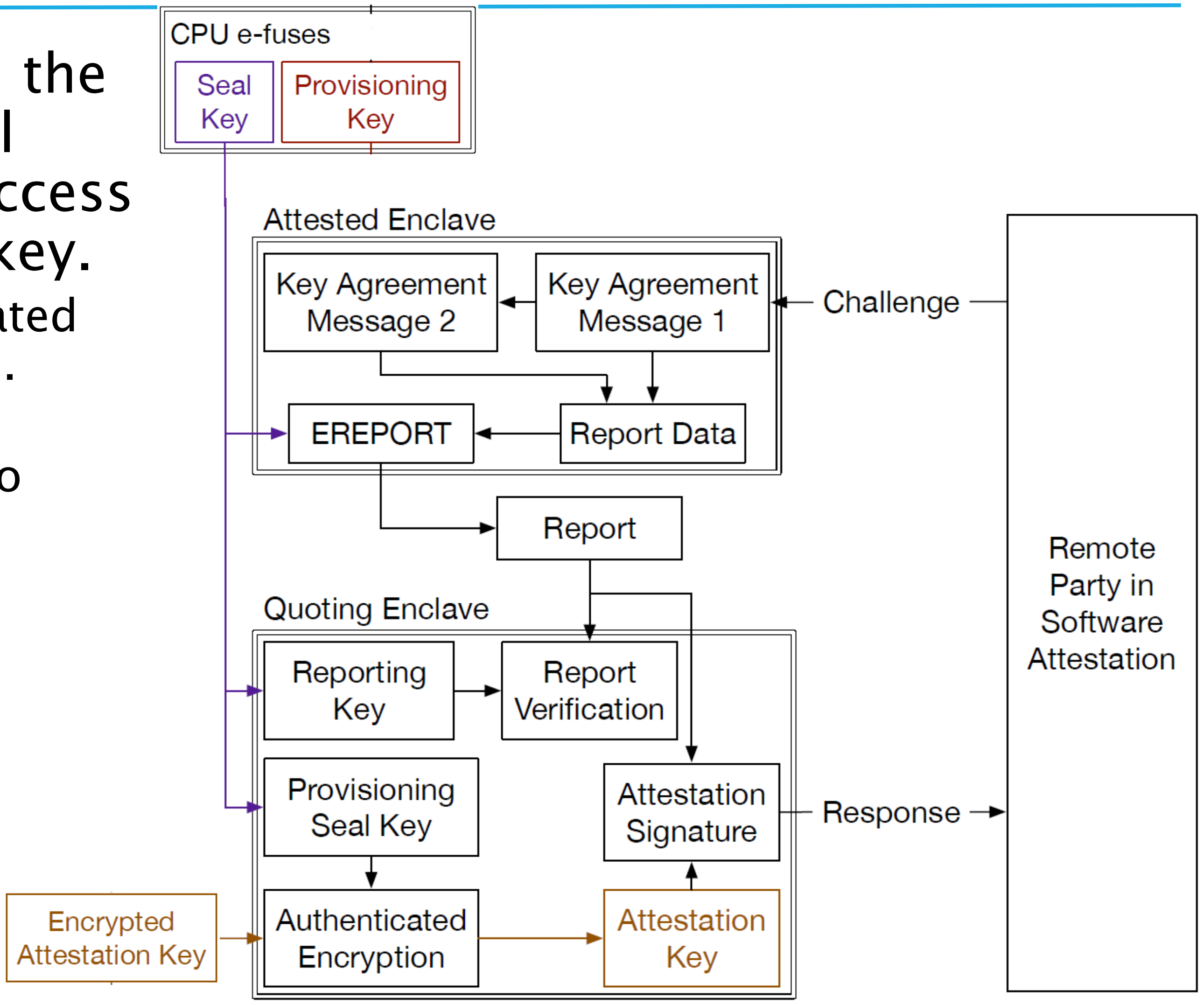


Outline

- SGX Memory Encryption Engine (MEE)
- SGX Memory Access Protection
- Tracking TLB Flushes
- Enclave Signature Verification
- SGX Security Properties
- Misconceptions about SGX
- Interaction with Anti-Virus Software

Misconceptions about SGX

- Remote attestation relies on the Quoting Enclave with special privileges that allows it to access the processor's attestation key.
  - This assumes the Enclave is isolated properly, but this is not true (e.g. cache side channel).
  - Intel suggests the programmer to remove data dependent memory access, especially for crypto algorithms.



Misconceptions about SGX

- Enclaves Can DOS (Denial-of-service) the System Software ❌
  - The SGX design provides system software the capability to protect itself from enclaves that engage in CPU hogging and DRAM hogging.
  - System software needs to reserve at least one LP for non-enclave computation.
- SGX is tamper-resistant ❌
  - The chip itself does not prevent physical tampering.

Outline

- SGX Memory Encryption Engine (MEE)
- SGX Memory Access Protection
- Tracking TLB Flushes
- Enclave Signature Verification
- SGX Security Properties
- Misconceptions about SGX
- Interaction with Anti-Virus Software

Interaction with Anti-Virus Software

- Today's anti-virus (AV) systems are pattern matchers.

- 1. A generic loader that is undetectable by AV's pattern matcher.
- 2. Load encrypted malicious payload from Internet.
- 3. Execute malicious code inside the Enclave. (botnets?)

- Possible solutions:
  - recording and filtering the I/O performed by software
  - Static analysis

References

- [1] Costan V, Devadas S. Intel sgx explained[R]. Cryptology ePrint Archive, Report 2016/086, 20 16. http://eprint. iacr. org.
- [2] SGX Tutorial on ISCA 2015.
- [3] Gueron S. Intel® Software Guard Extensions (Intel® SGX) Memory Encryption Engine (MEE), RWC 2016.
- [4] Gueron S. A Memory Encryption Engine Suitable for General Purpose Processors[J].
- [5] Bernstein D J. Stronger security bounds for Wegman-Carter-Shoup authenticators[M]//Advances in Cryptology-EUROCRYPT 2005. Springer Berlin Heidelberg, 2005: 164-180.