Twentyfirst Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)

September 4th, 2024

Fault Tolerance of Encrypted Memory: Crash Consistency Problem and Secure Recovery

Rei Ueno

Kyoto University, Japan

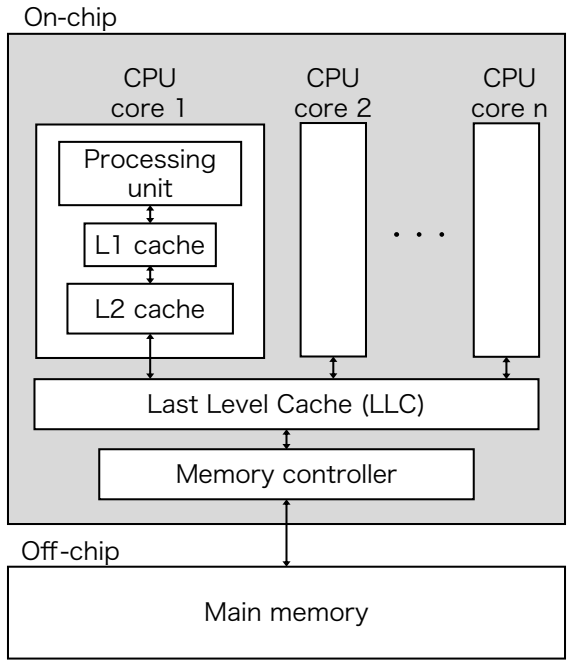
京都大学

KYOTO UNIVERSITY

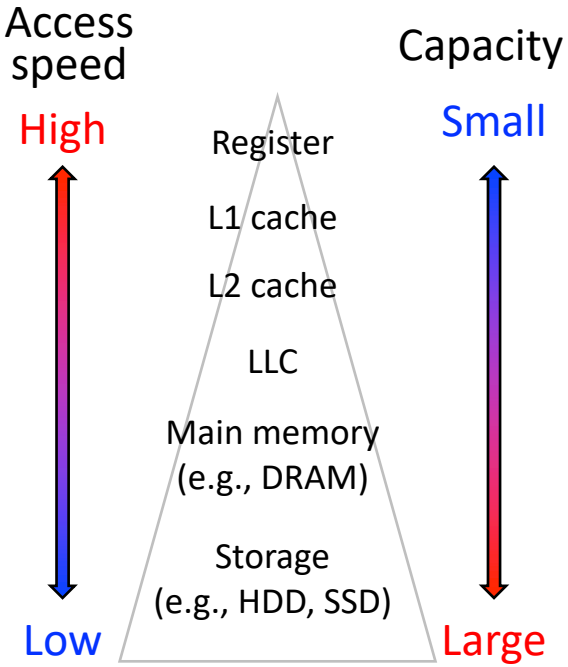
This talk is based on collaborative works with Maya Oda, Hiromichi Haneda, Naofumi Homma (Tohoku University) and Akiko Inoue, Kazuhiko Minematsu (NEC)

Modern computers and memory

- Modern CPUs operate with high frequency (~6 GHz), while memory access speed is relatively low and can be performance bottleneck
- Cache memories on-chip decrease frequency of main memory accesses
  - Memory access speed and capacity are tradeoff
  - Off-chip memory is essential to handle practical amount of data
- Off-chip memory is subject to eaves-dropping and manipulation
  - Motivation of memory encryption



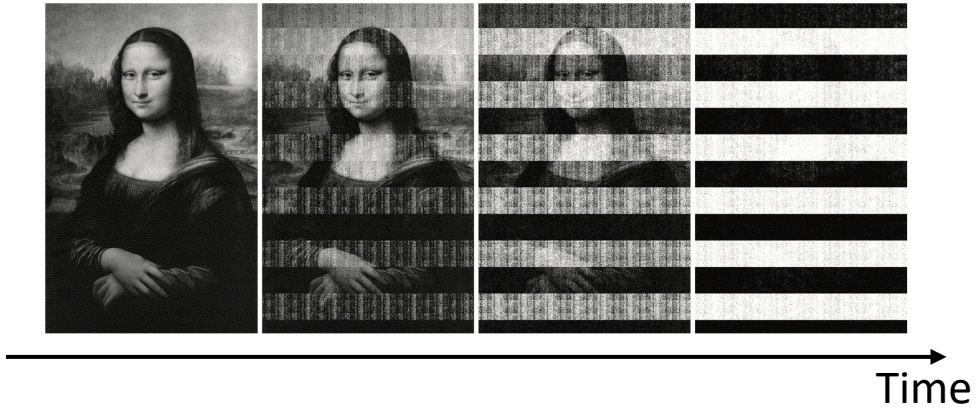
Typical system model of modern computers



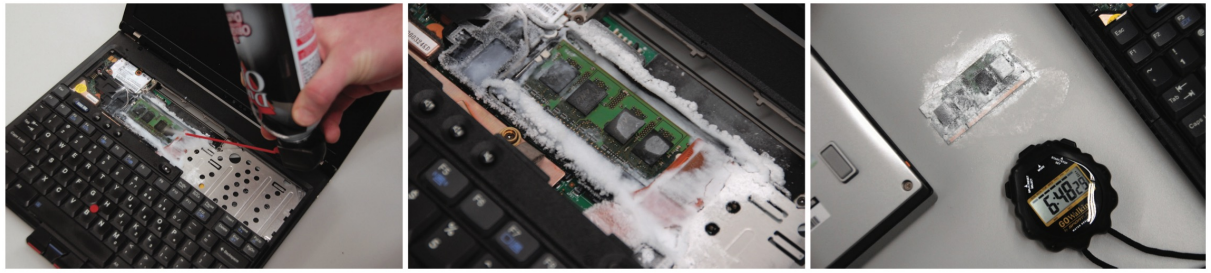
Memory hierarchy

Cold boot attack [HSH+09]

- Main memory, typically DRAM, holds data by capacitor charge
  - Memory data remains for a time but is not erased immediately after powered-off
  - Cooling DRAM makes the data remaining time longer
- Memory data can be eavesdropped by dumping it after removing
  - Allows for bypassing storage encryption to retrieve confidential data
  - Successful key recovery of RSA, AES, lattice-based crypto, etc. has been reported



Time elapse of powered-off DRAM data

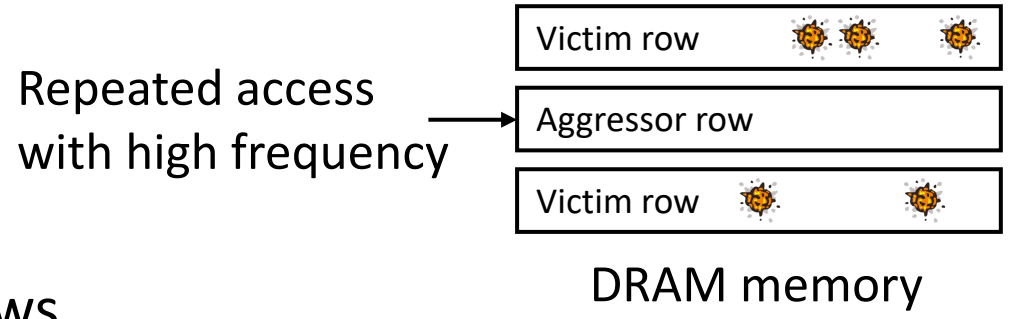


Experiment of cooling and removing DRAM module

[HSH+09] J. A. Halderman et al., "Lest we remember: cold-boot attacks on encryption keys," *Communication of the ACM*, vol. 52, pp. 91–98, 2009. Figures are from the article.

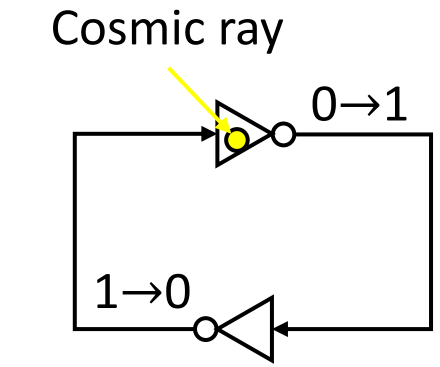
Threats on memory data integrity

- Rowhammer [KDK+14]
  - Incur bit flips without accessing themselves
  - High-frequent access to aggressor affects electrical charges of capacitors in neighbor rows
  - Performed remotely, resulting in some serious vulnerability



DRAM memory

- Soft-error
  - Bits in DRAM and SRAM are sometimes flipped accidentally
    - Because of cosmic rays reaching earth surface, noise, and unexpectedly large leak of electrical charge, etc.
  - More serious for large memory and cutting-edge technologies



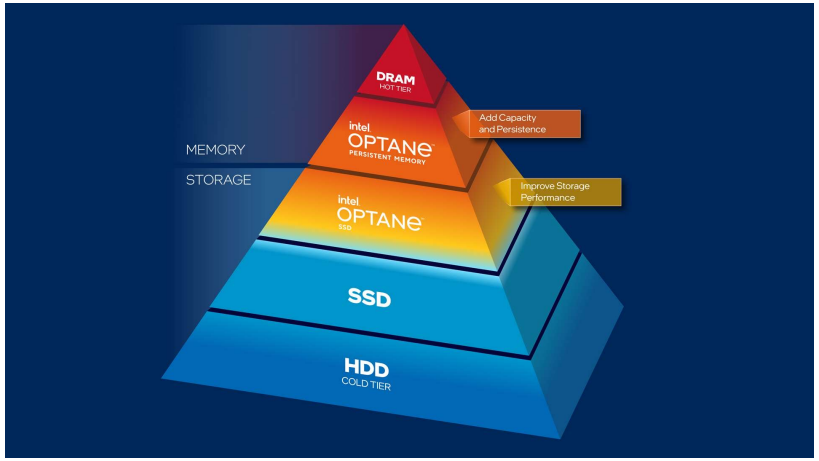
Inverter ring in SRAM

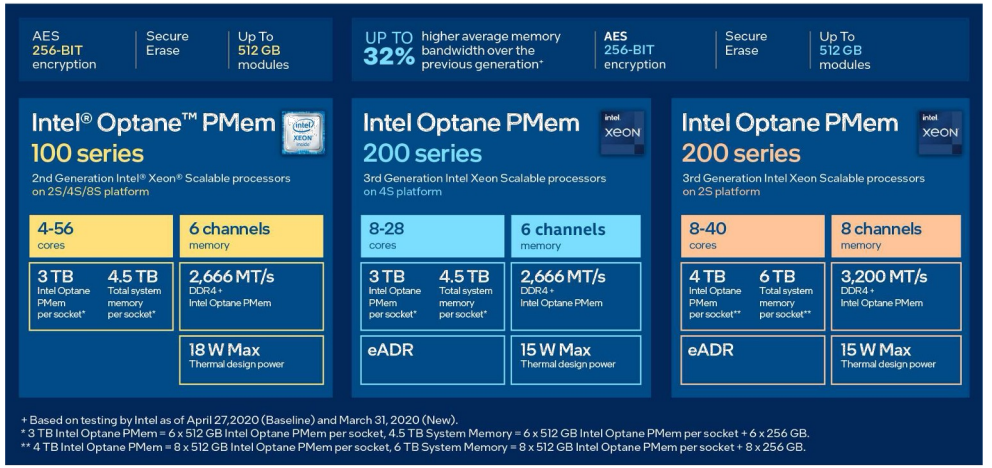
- And other threats, such as abuse of privilege, compromised OS/hypervisor, microarchitectural side-channel attacks, etc.

[KDK+14] Y. Kim et al., "Flipping bits in memory without accessing them," *ISCA*, 2014.

Non-volatile memory (NVM) for main memory

- NV main memory contributes to high performance and low energy CPU
  - Intel Optane Persistent Memory, NVDIMM, etc.
- Actively researched in microarchitectural field (e.g., MICRO, ISCA, HPCA)
- Eavesdropping and manipulation are more severe and practical for NVM
  - Intel Optane Persistent Memory utilizes AES–XTS engine for confidentiality





Overview of Intel Optane Persistent Memory

https://www.intel.co.jp/content/www/jp/ja/products/docs/memory-storage/optane-persistent-memory/optane-persistent-memory-200-series-brief.html

Security levels of memory encryption

- Memory encryption can be classified into three levels [AMS+22]
  - Major goals are confidentiality, authenticity, and resistance to replay attacks
  - Replay attack prevention needs special care specific to memory encryption

	Level 1 (L1)	Level 2 (L2)	Level 3 (L3)
Goals	Confidentiality	Confidentiality Integrity	Confidentiality Integrity Replay protection
Major primitives	Encryption	Encryption (Nonce-based) MAC/AE	Encryption Nonce-based MAC/AE Memory authentication tree
Examples	AES–XTS for storage encryption and Intel Optane	Intel TDX, ARM SEV	Intel SGX

[AMS+22] R. Avanzi et al., "SoK: Hardware-Supported Cryptographic Protection of Random Access Memory," *IACR ePrint Archives*, 2022/1472

Dilemma between authenticity and availability

- Computer systems sometimes crash
  - Sudden power-off, blackout, fluctuation of power supply, etc.
- Bits in memory (DRAM, SRAM, and NVM) are sometimes flipped
  - Server-grade memory uses ECC (Single Error Correction and Double Error Detection)
  - Consumer-grade ones (e.g., laptop and smartphone) do not
- MAC/AE detects even one-bit modification for simple security goal
  - If bits are accidentally or maliciously flipped, whole memory data gets unavailable
  - Memory encryption degrades data availability against soft-errors and DoS-like attacks
  - Users and vendors would be very sensitive to data availability
- Level 4 (L4): Memory recoverability against errors and system crashes
  in addition to confidentiality, authenticity, and resilience to replay attacks

This talk

- Overview and basics of memory encryption
  - Threat and system models, major cryptographic primitives, hardware architecture
  - Focus on L3-secure memory encryption
- Fault tolerance/crash consistency problem of encrypted memory
  - Consider malicious manipulation, soft-errors, and system crashes
  - Secure recovery is non-trivial problem
- Crystalor: State-of-the-art L4-secure memory encryption mechanism [UHH+24]
  - Almost no latency overhead recovery mechanism in CPU performance based on tailor-made primitive
  - Offer fast recovery after error detection or system crash

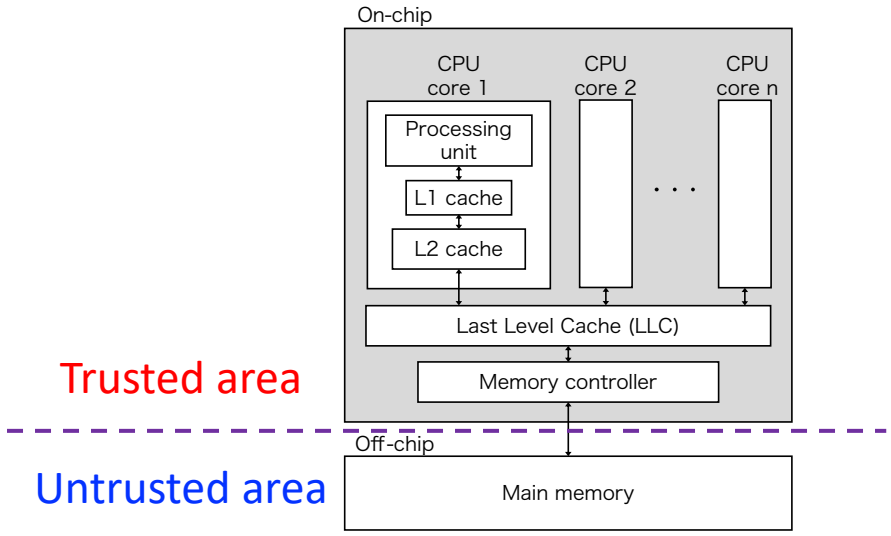
[UHH+24] R. Ueno, H. Haneda, N. Homma, A. Inoue, K. Minematsu, “Crystalor: Recoverable memory encryption mechanism with Optimized Metadata Structure,” *ACM CCS*, 2024. (to appear)

Talk outline

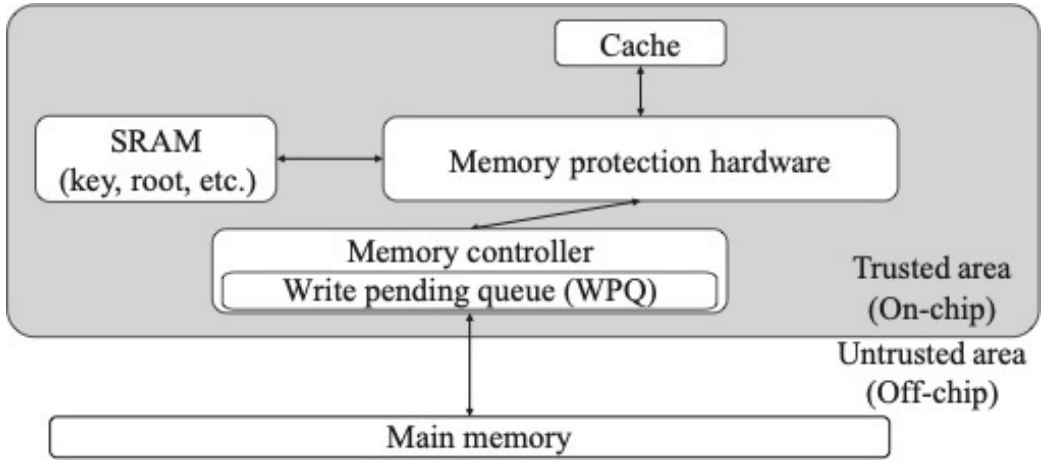
- Background
- Overview and basics of memory encryption
- Crash consistency problem
- Crash recovery mechanisms for encrypted memory
- Concluding remarks

Threat and system models of memory encryption

- System is divided into on-chip trusted and off-chip untrusted areas
  - Attacker can neither eavesdrop nor manipulate on-chip data
    - Do not consider on-chip side-channel attacks (protected by different means)
    - Limited capacity of memory (NVM), basically just for secret keys and root
  - Attacker can perform arbitrary eavesdropping and manipulation on off-chip data
    - Subject to confidentiality and authenticity



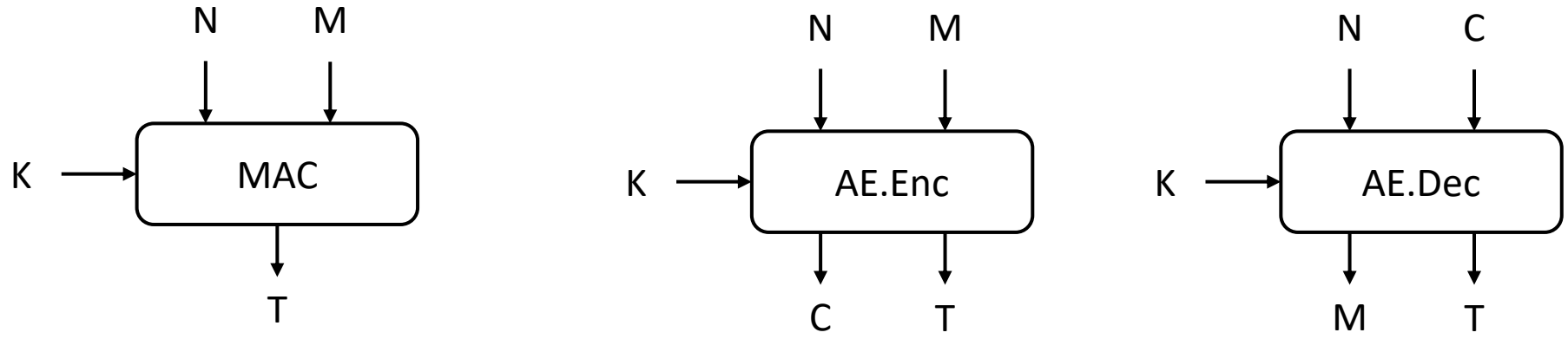
Typical system model of modern computers



Major components related to memory encryption

Cryptographic primitives for memory encryption

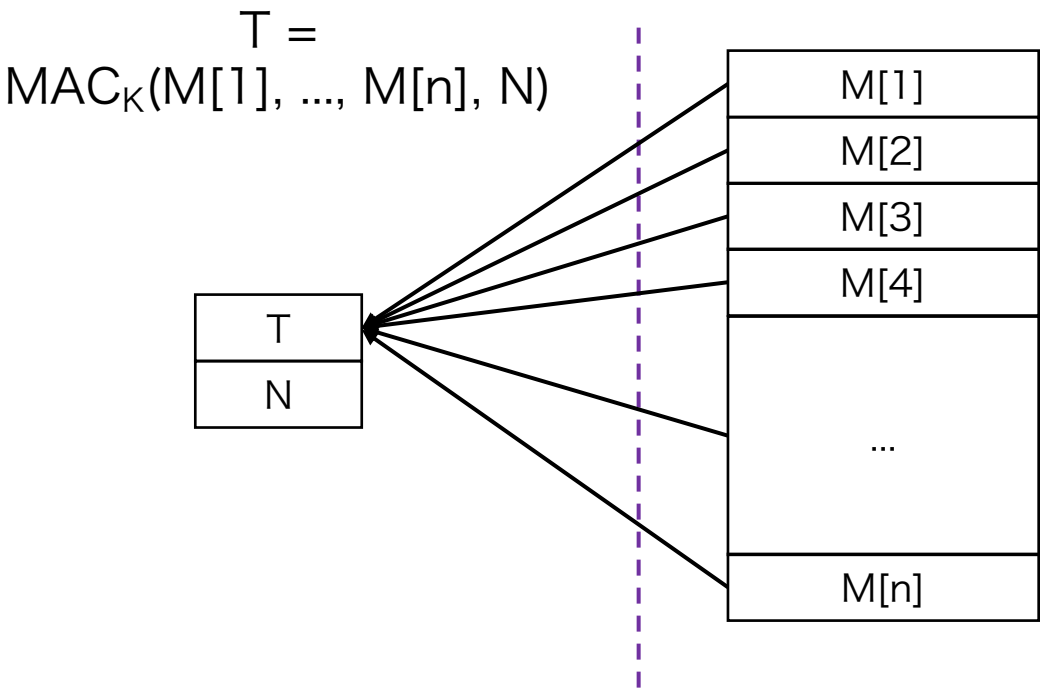
- Symmetric ciphers are usually used
  - On-chip memory encryption engine performs both encryption and decryption when writing and reading data to/from memory, respectively
  - No need of key exchange, and the secret key never exposes outside chip
- Confidentiality and authenticity for memory data
  - Nonce-based message authentication code (MAC) and authenticated encryption (AE)



K: Secret key, N: Nonce, M: Message/Plaintext, C: Encrypted data/Ciphertext, T: Tag

Memory encryption for integrity: Attempt 1/3

- Simple calculation of one MAC tag for full memory data
  - Need access to full domain of the memory (GB or TB order)
    - Impossible to compute the tag in real-time during nominal operation

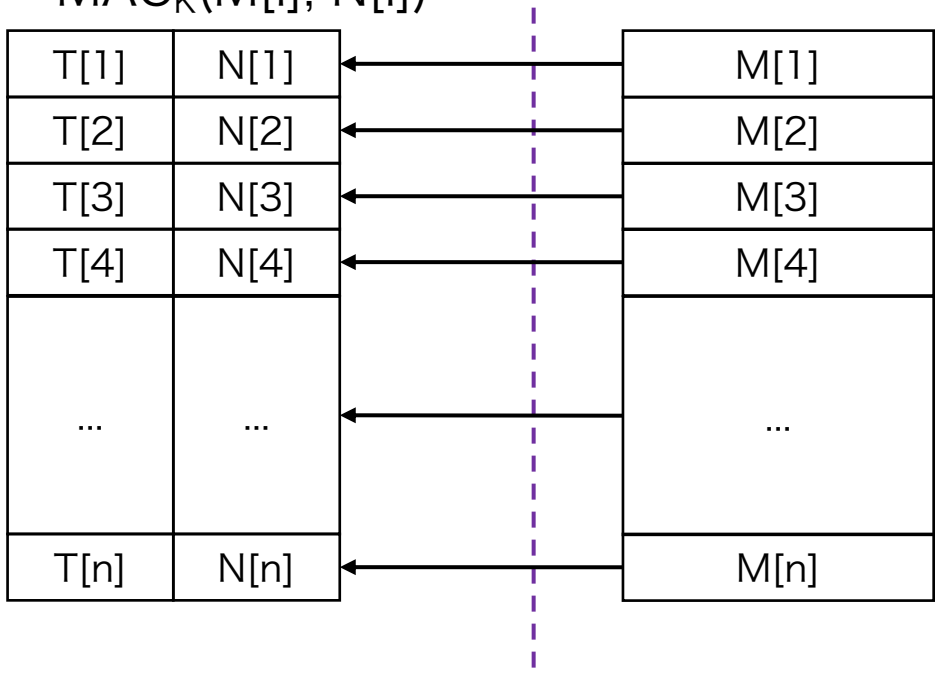


Security	Level 3
Computational cost at one store/read	O(n)
On-chip memory	O(1)
Off-chip memory	No overhead

Memory encryption for integrity: Attempt 2/3

- Calculate MAC tag for each data block
  - Need on-chip NVM for storing tags and nonces as large as off-chip memory
    - On-chip memory is limited, and storing large contents on-chip is meaningless

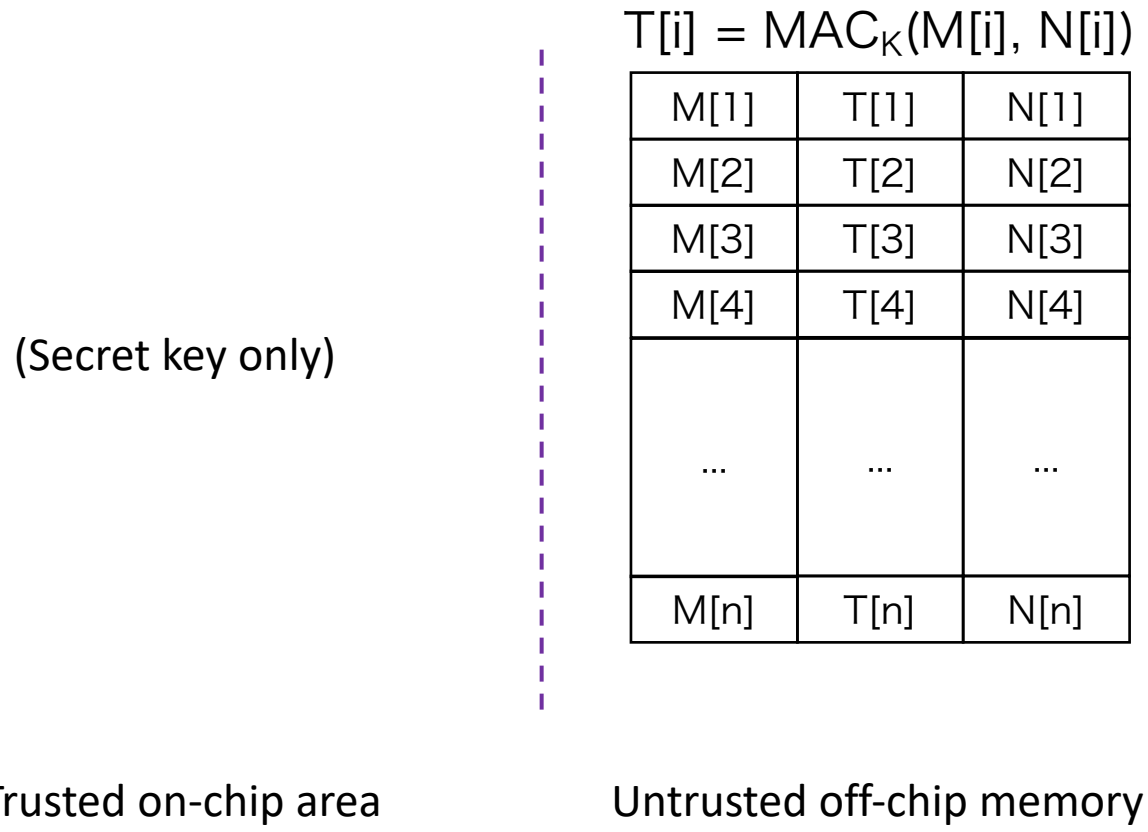
$T[i] = MAC_K(M[i], N[i])$



Security	Level 3
Computational cost at one store/read	O(1)
On-chip memory	O(n)
Off-chip memory	No overhead

Memory encryption for integrity: Attempt 3/3

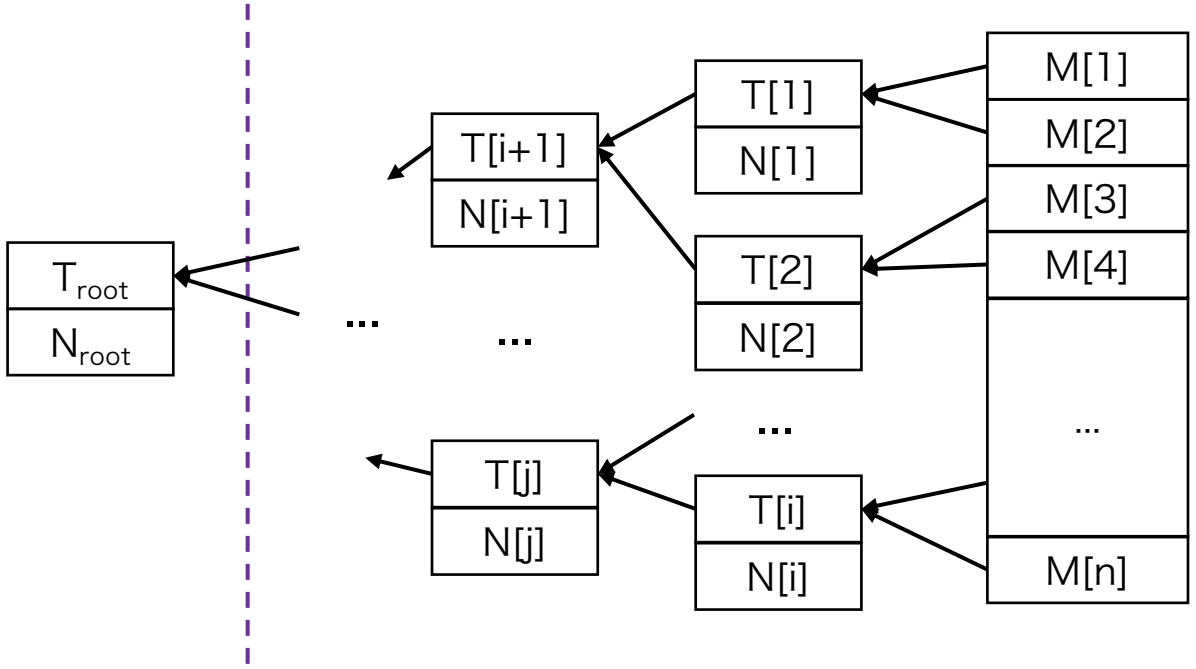
- Store pair of nonce and computed tag on-chip (L2 scheme)
  - Cannot prevent replay attacks
    - Verification of any memory data should use root-of-trust tag/nonce on-chip



Security	Level 2 (Vulnerable to replay attacks)
Computational cost at one store/read	O(1)
On-chip memory	O(1) Secret key only
Off-chip memory	O(n)

Practical solution: Memory authentication tree

- Realize real-time encryption/decryption and authentication
  - When accessing an address, verify only its related tags
- Realize replay attack protection with fixed on-chip memory overhead

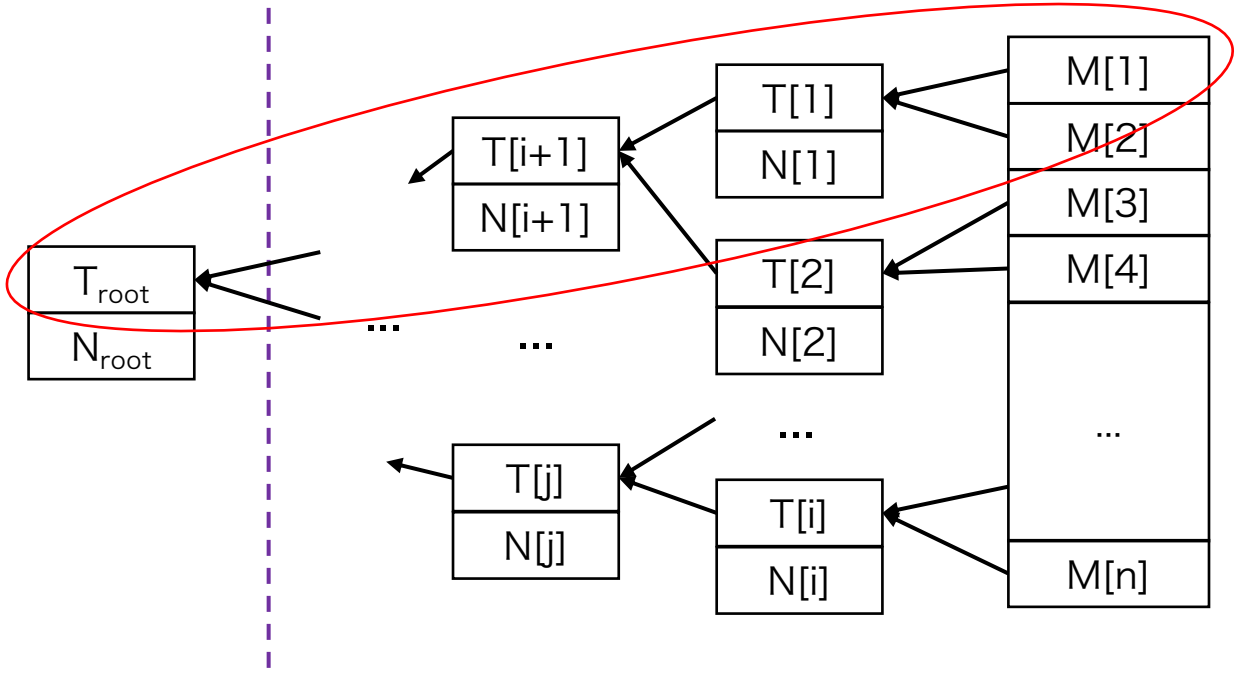


Security	Level 3
Computational cost at one store/read	$O(bd) = O(d \log_d n)$
On-chip memory	O(1)
Off-chip memory	$O(b^{d-1})$

d: tree depth, b: tree arity

Practical solution: Memory authentication tree

- Realize real-time encryption/decryption and authentication
  - When accessing an address, verify only its related tags
- Realize replay attack protection with fixed on-chip memory overhead

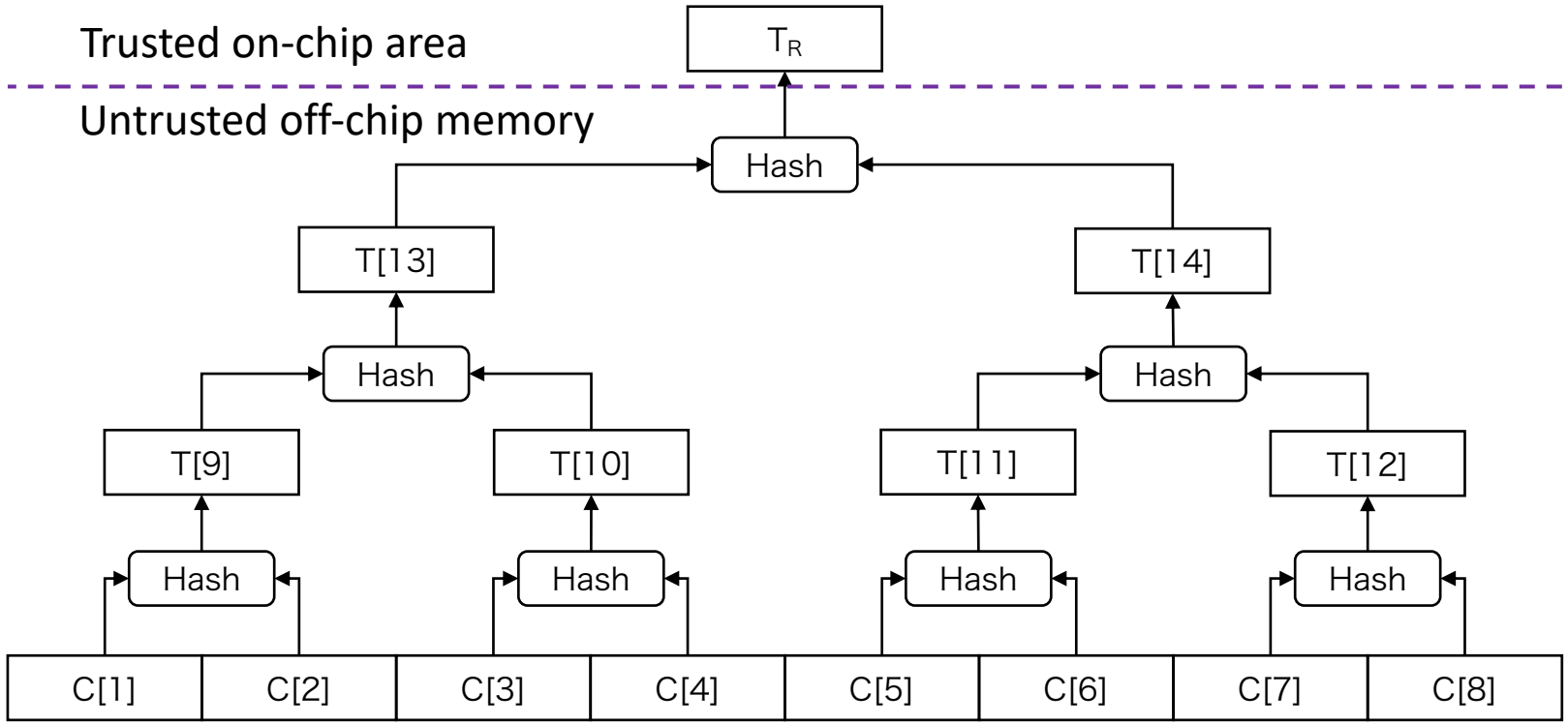


Security	Level 3
Computational cost at one store/read	$O(bd) = O(d \log_d n)$
On-chip memory	O(1)
Off-chip memory	$O(b^{d-1})$

d: tree depth, b: tree arity

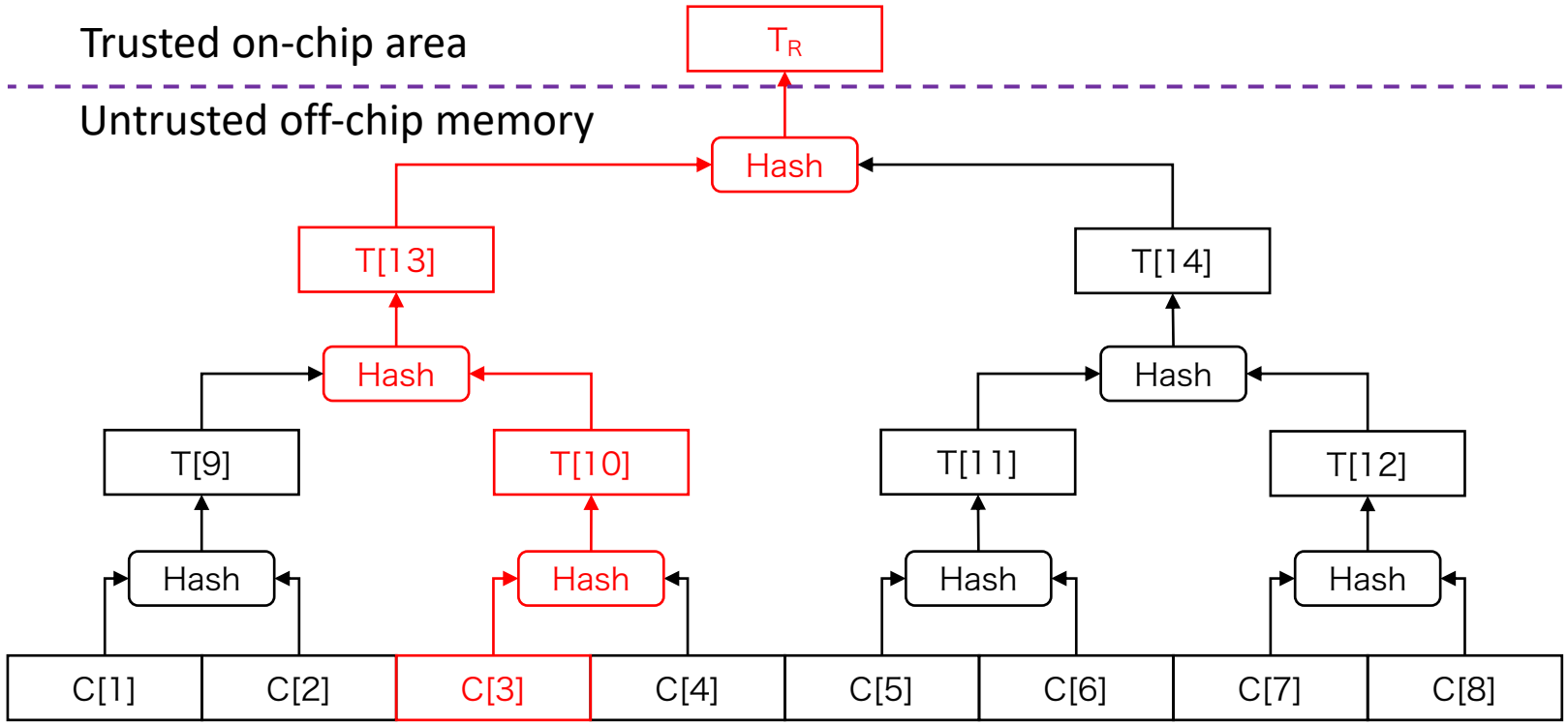
Merkle tree (MT)

- First and most major authentication tree construction
  - Frequently used in cryptography (e.g., hash-based signature)
  - Parent node verifies children's tag and root tag is securely stored on-chip
  - HMAC–SHA1/2/3 and AES–CTR are commonly used



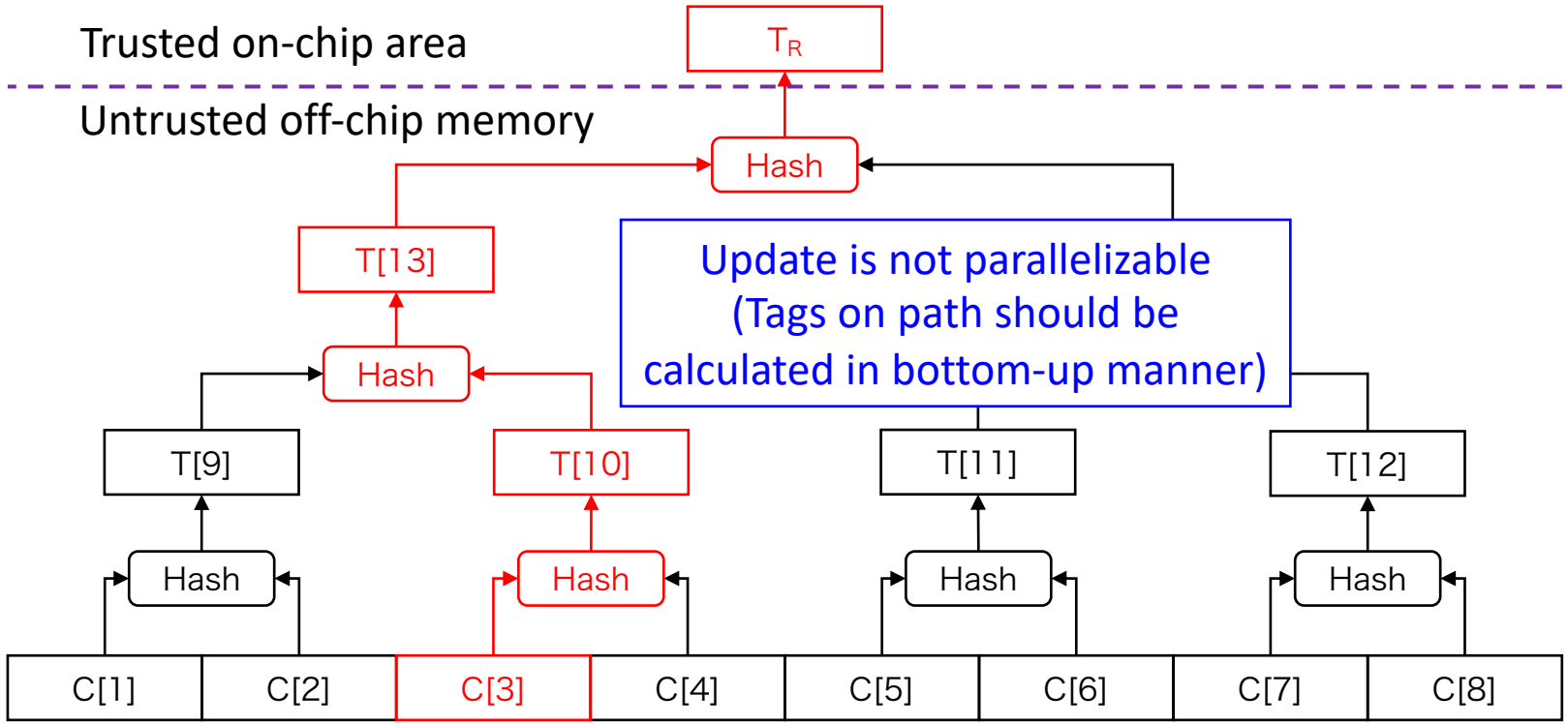
Merkle tree (MT)

- First and most major authentication tree construction
  - Frequently used in cryptography (e.g., hash-based signature)
  - Parent node verifies children's tag and root tag is securely stored on-chip
  - HMAC–SHA1/2/3 and AES–CTR are commonly used



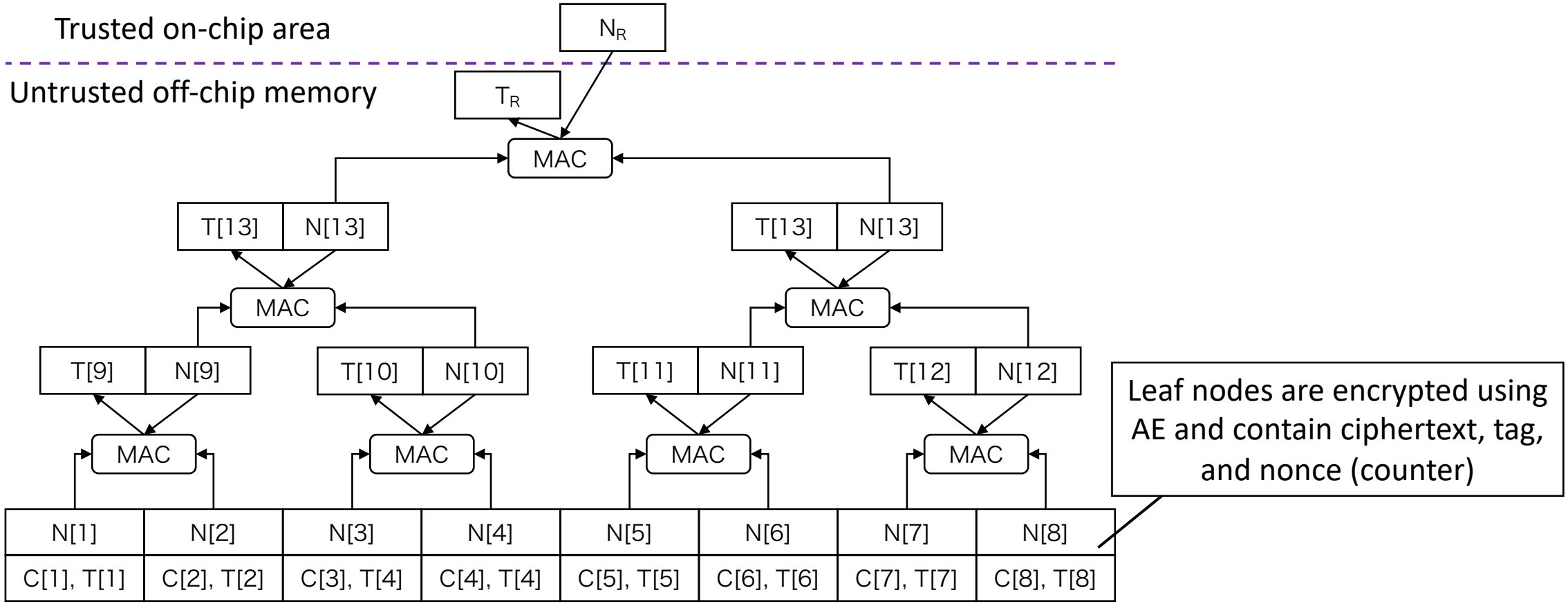
Merkle tree (MT)

- First and most major authentication tree construction
  - Frequently used in cryptography (e.g., hash-based signature)
  - Parent node verifies children's tag and root tag is securely stored on-chip
  - HMAC–SHA1/2/3 and AES–CTR are commonly used



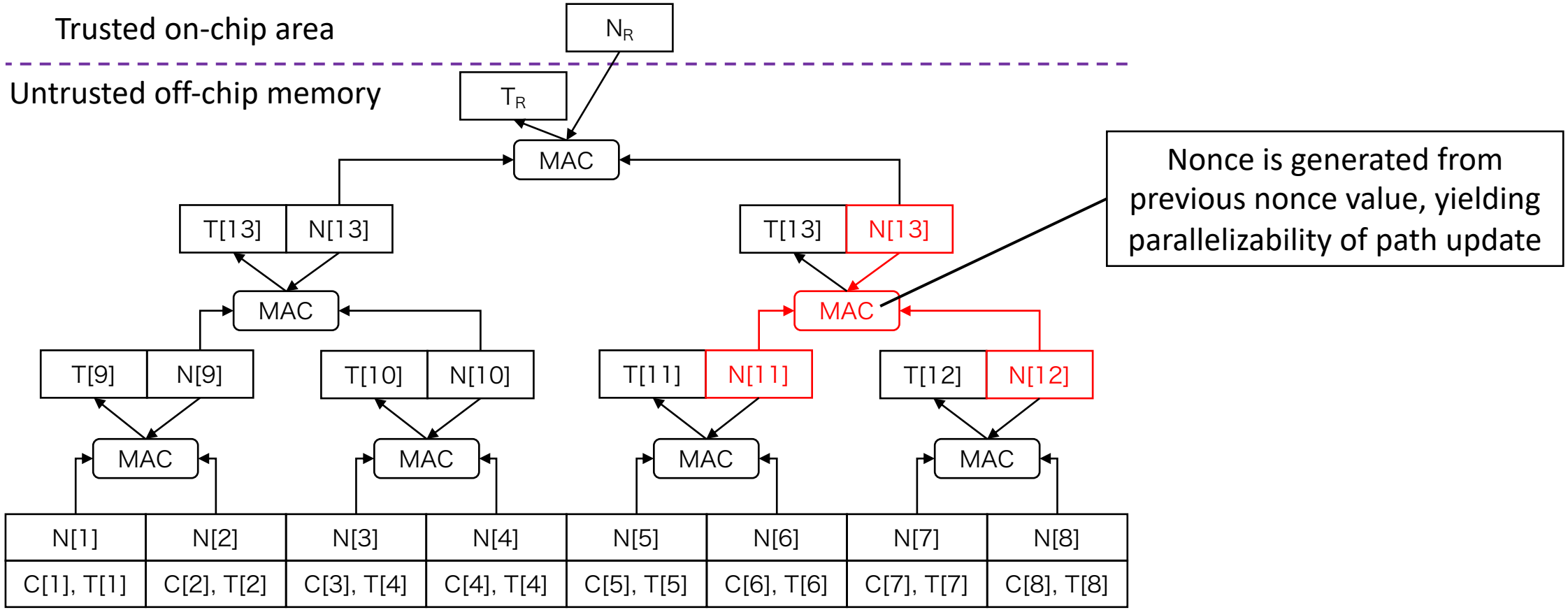
Parallelizable authentication tree (PAT)

- Parent nodes verifies children's **nonces** instead of tags
  - Update is parallelizable, because new nonce is generated without children's data



Parallelizable authentication tree (PAT)

- Parent nodes verifies children's **nonces** instead of tags
  - Update is parallelizable, because new nonce is generated without children's data



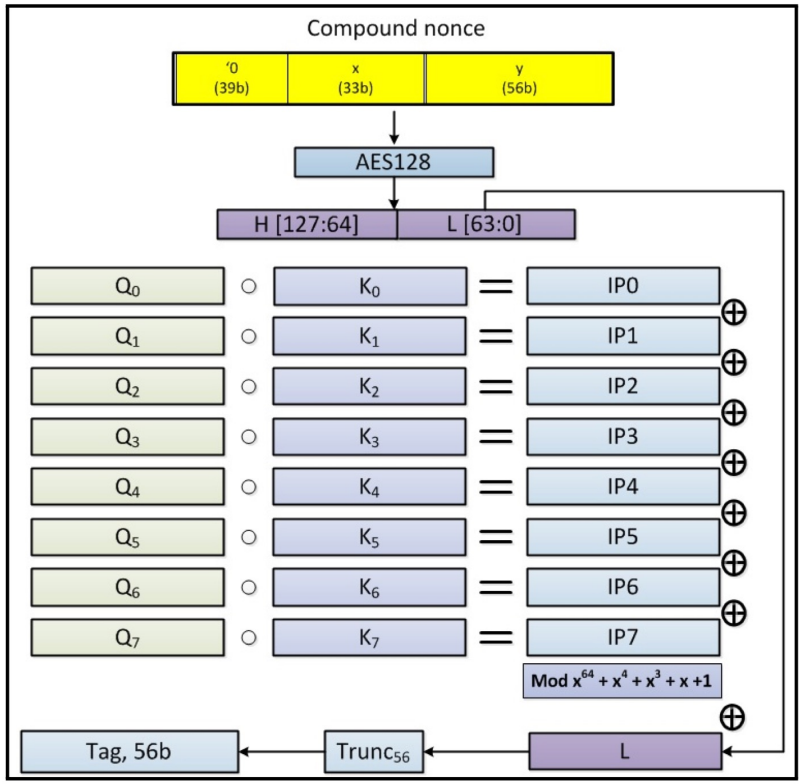
Comparison of MT and PAT in memory encryption

- MT has critical drawback of non-parallelizability of path update
  - Non-negligible latency overhead of write to memory
  - More critical when deeper tree for protecting large memory
- PAT has algorithmically lower latency than MT, yielding higher performance
  - Promising for memory encryption, if we can realize fault tolerance with low cost

	MT	PAT
Verification (Read from memory)	Parallelizable	Parallelizable
Update (Write to memory)	Not parallelizable	Parallelizable
Fault tolerance	Relatively easy	Non-trivial

PAT instance 1: Intel SGX Integrity Tree (SIT) [Gue16]

- Major memory encryption scheme based on PAT
  - Studied also in microarchitecture conferences
  - Generic composition of AES–CTR and Wegman–Carter MAC with 56-bit tag
    - Security reduction to AES-128
    - The MAC uses universal hash over GF(2^{64})
      - Authenticate 512-bit data using 512-bit key
      - Sum of [64-bit data chunk]×[64-bit key chunk] (i.e., inner-product over GF(2^{64}))
      - Information-theoretical security
- Specialized to protect 128 MB region
  - But difficult in scaling to larger region

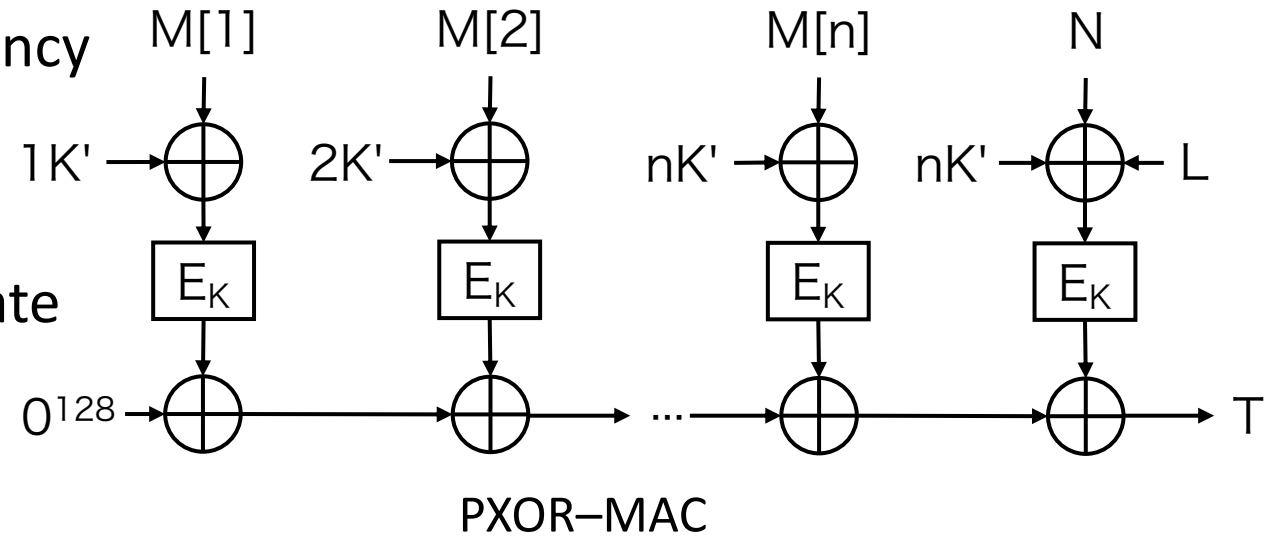


Wegman–Carter MAC in SIT

[Gue16] S. Gueron, "A Memory Encryption Engine Suitable for General Purpose Processors," *IACR ePrint Archive*, 2016/204. Figure is from the paper.

PAT instance 2: Encryption for Large Memory (ELM) [IMO+22]

- Develop AE and MAC specialized to PAT-based memory encryption
  - PXOR–MAC: Optimized PMAC with block cipher (BC) depth 1
  - Flat-OCB (AE): Optimized OCB with lowest BC depths in both enc. and dec.
- Especially efficient in protecting larger region
  - Rate-1, parallelizability, and minimum depth achieve optimal AE/MAC latency
  - Required on-chip NVM size is fixed in contrast to SIT
  - PXOR–MAC offers incremental update
    - Further efficient memory write



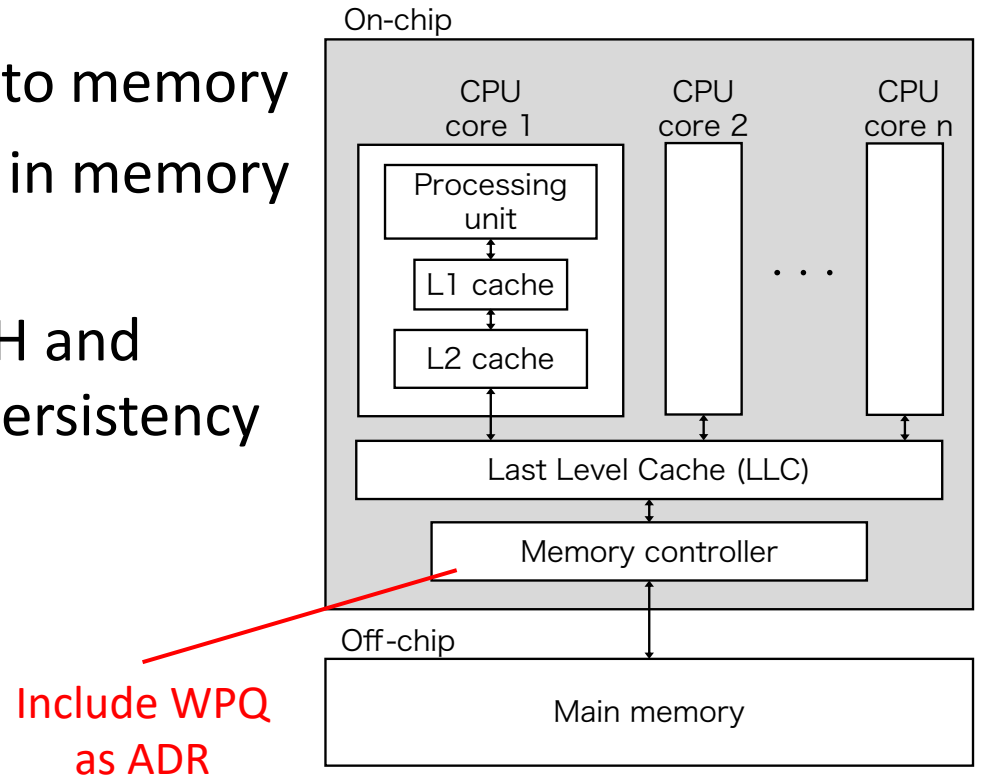
[IMO+22] A. Inoue, K. Minematsu, M. Oda, R. Ueno, N. Homma, "ELM: A Low-Latency and Scalable Memory Encryption Scheme," *IEEE Transactions on Information Forensics and Security*, 17, pp. 2628–2643, 2022.

Talk outline

- Background
- Overview of memory encryption
- Crash consistency problem
- Crash recovery mechanisms for encrypted memory
- Concluding remarks

Crash consistency and data persistency

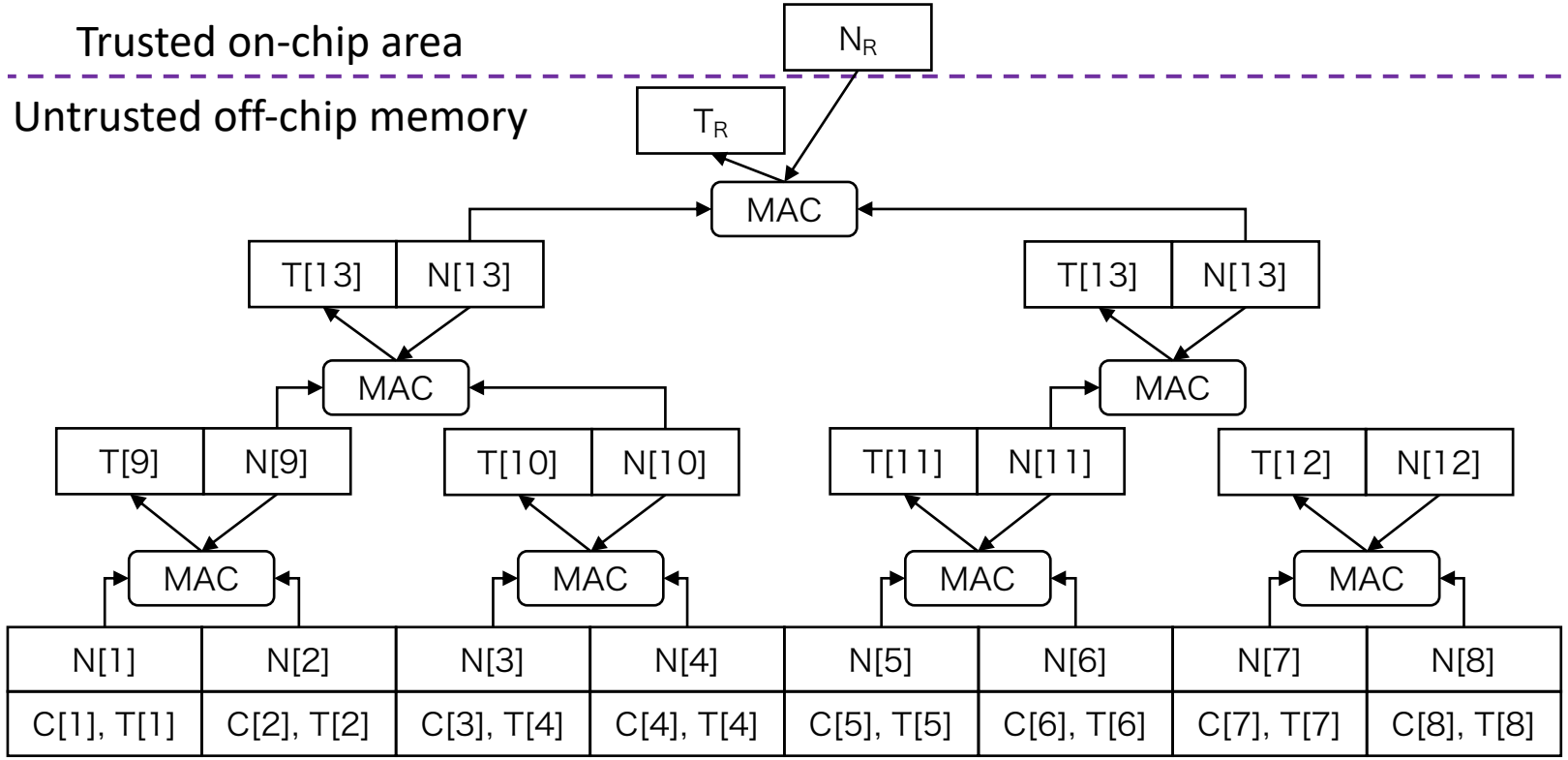
- Crash includes sudden power-off/blackout, fluctuation of power supply, etc.
- Need to persist data to be written to main memory
- Asynchronous DRAM refresh (ADR) domain
  - Data reached here is guaranteed to be written to memory
  - Some CPUs utilize write pending queue (WPQ) in memory controller as ADR
  - At cache miss or some instructions like CLFLUSH and CLWB, cache data should reach to ADR for its persistency (for write-back policy)
  - Need some care to avoid persistency bugs (especially for write-through policy)



System model of modern computers

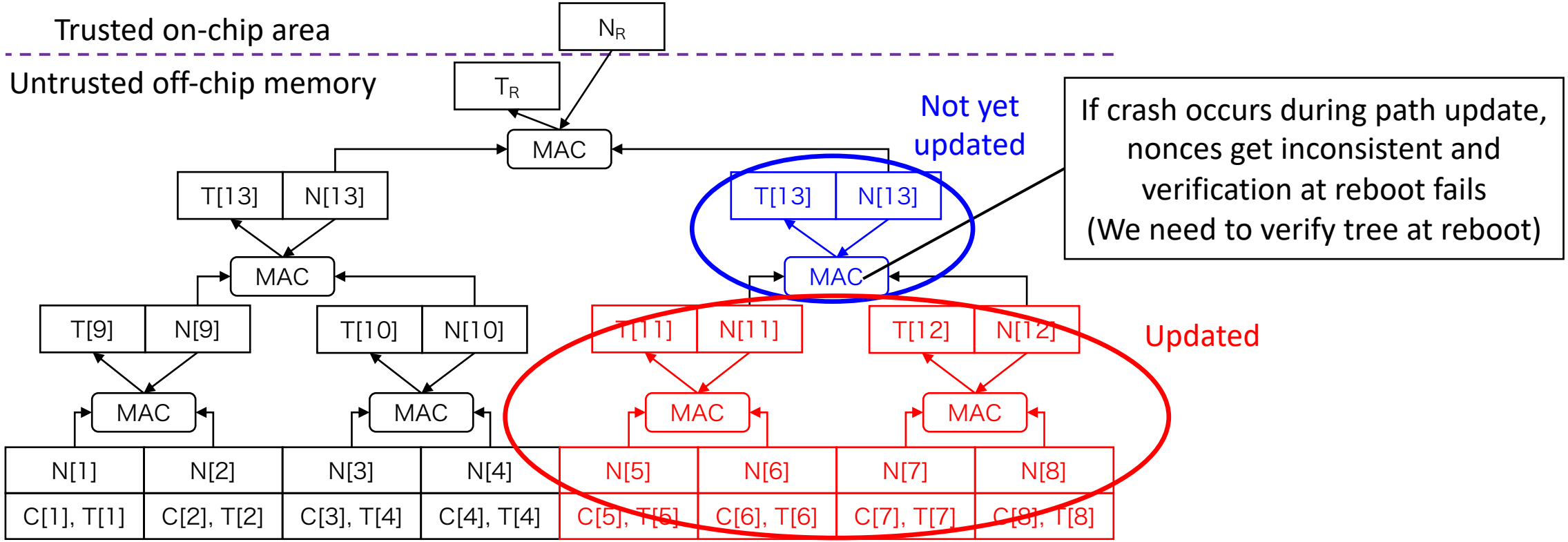
Crash consistency of encrypted memory

- At memory write, path is updated serially due to limited bandwidth
  - Although PAT path update is parallelizable in principle
  - Timing when some nodes are updated but others are not must exist
  - If verification of a node fails, related data get unavailable as it may be attacked



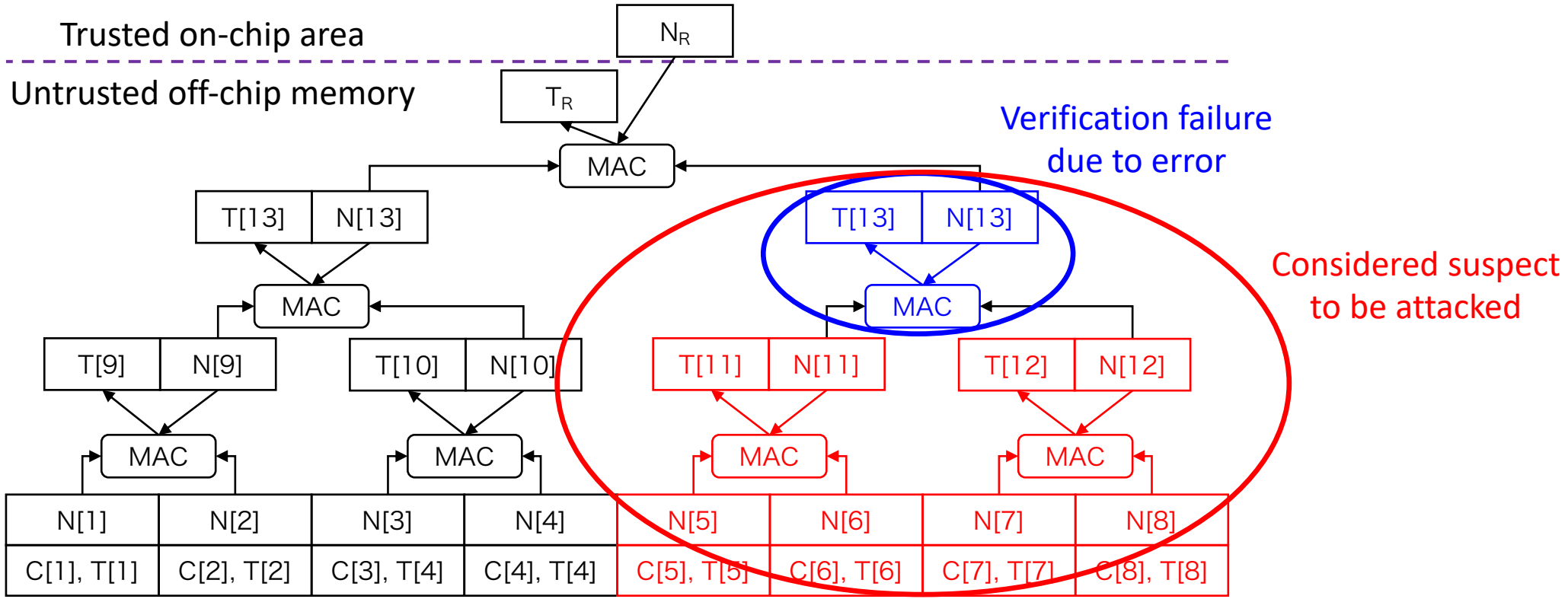
Crash consistency of encrypted memory

- At memory write, path is updated serially due to limited bandwidth
  - Although PAT path update is parallelizable in principle
  - Timing when some nodes are updated but others do not must exist
  - If verification of a node fails, related data get unavailable as it may be attacked



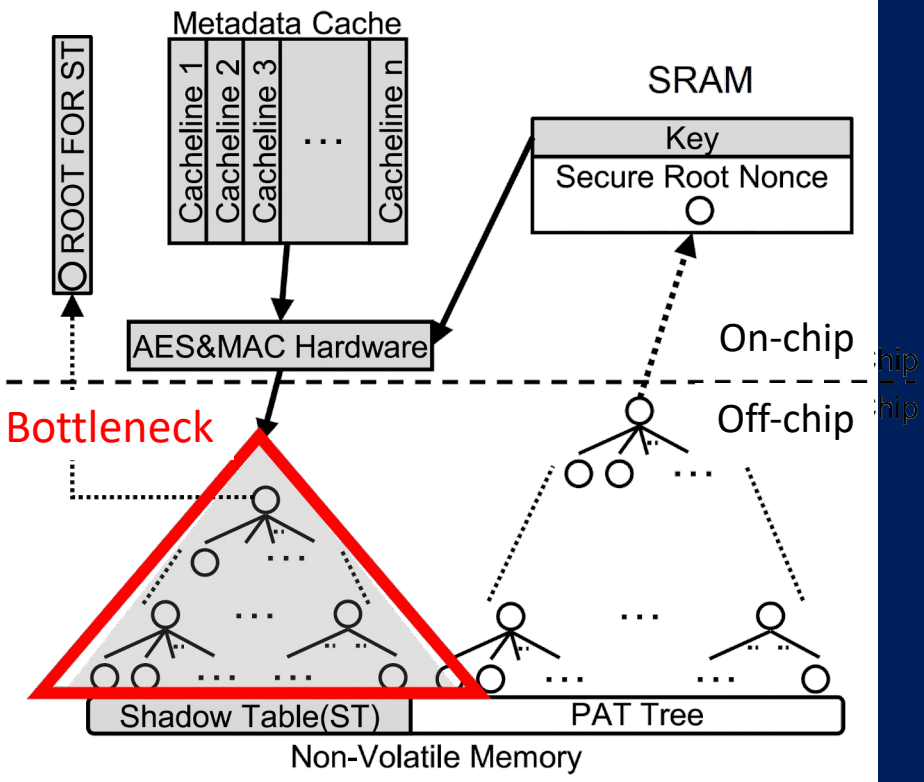
Errors in encrypted memory

- Security metadata for PAT (intermediate nodes) are also subject to errors
  - Even one-bit error results in verification error, rendering related data unavailable
- **Memory encryption renders memory fault tolerance very difficult**



Existing crash consistency mechanism 1/2: Anubis [ZA19]

- Store information on nodes to be updated in *Shadow Table (ST)* at memory
  - At crash, recover the nonces and tags according to ST
  - Offer *lazy recovery*, yielding very fast recover
    - Anubis can guarantee integrity without full verification of whole memory data
- Need non-negligible latency overhead
  - ST forms MT, PAT is unavailable for ST to resolve PAT's problem
  - Contaminate PAT's advantage
  - Degrade CPU performance non-negligibly (yet most efficient when its publication)

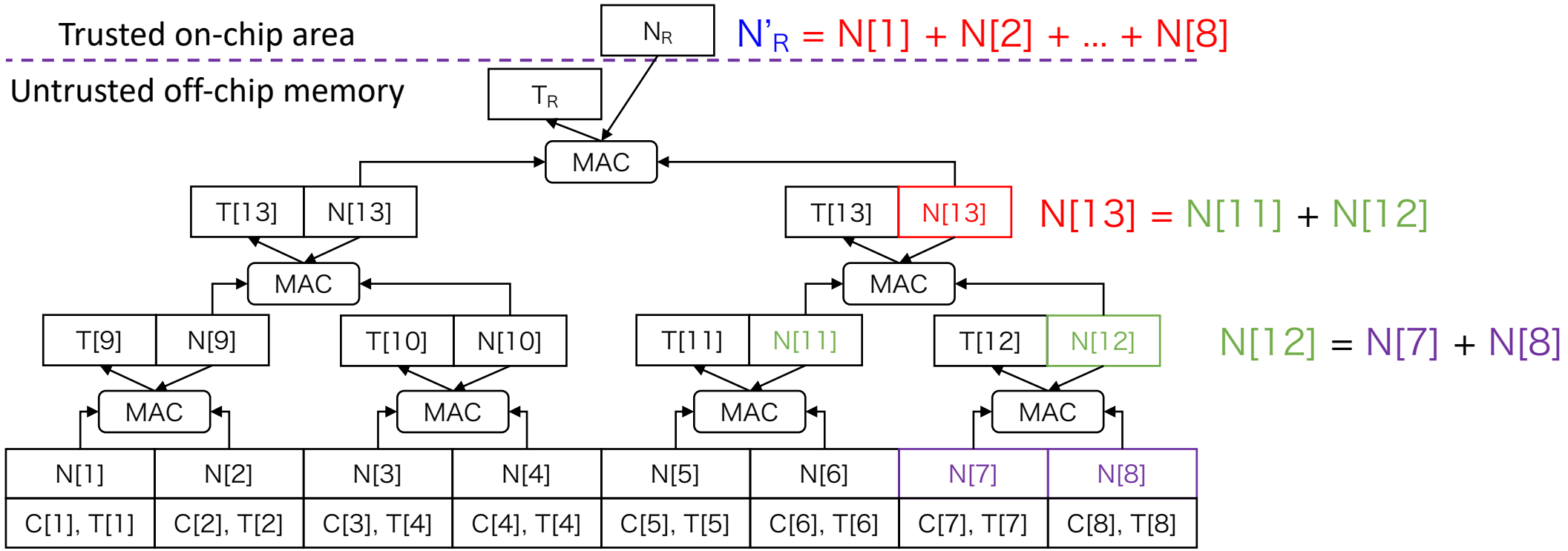


Overview of Anubis

[ZA19] K.A. Zubair, A. Award, "Anubis: ultra low-latency overhead and recovery time for non-volatile memories," *ISCA*, 2019.

Existing crash consistency mechanism 2/2: SCUE [HH23]

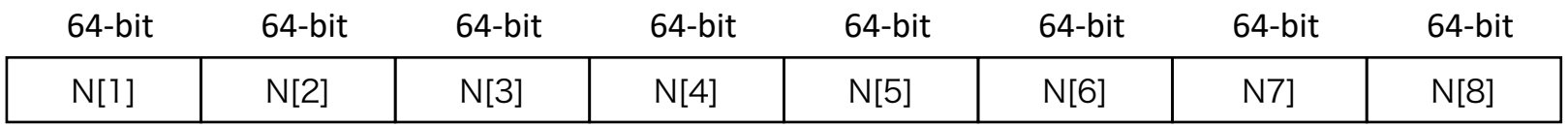
- Assume that nonce is given by upcounter, representing # updates of node
  - # Parent node updates equal to sum of # children nodes updates
    - Use distinct root nonce to recovery, representing sum of leaf counters
    - At crash or verification error, recover from leaves to upper nodes



[HH23] J. Huang, Y Hua, "Root Crash Consistency of SGX-style Integrity Trees in Secure Non-Volatile Memory Systems," *HPCA*, 2023.

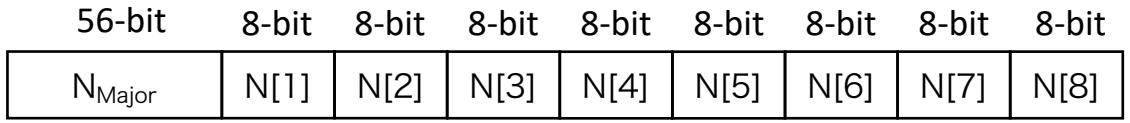
Drawback and limitation of SCUE

- Nonces should be realized by **just simple upcounter** for SCUE adoption
- In practice, optimization to compress tree is adopted, e.g. Split Counter (SC)
  - Major tree-structural optimization mechanism to compress security metadata [YEP+06]
  - Shares upper bits of counters by multiple nodes
  - Yield significant reduction of memory overhead, saving bandwidth and latency
- SCUE is not applicable to SC-based PAT, which is significant disadvantage



Nonces of 8 nodes are **512 bits in total**

Plain nonce counter: i-th node counter is simply given by 64-bit N[i]



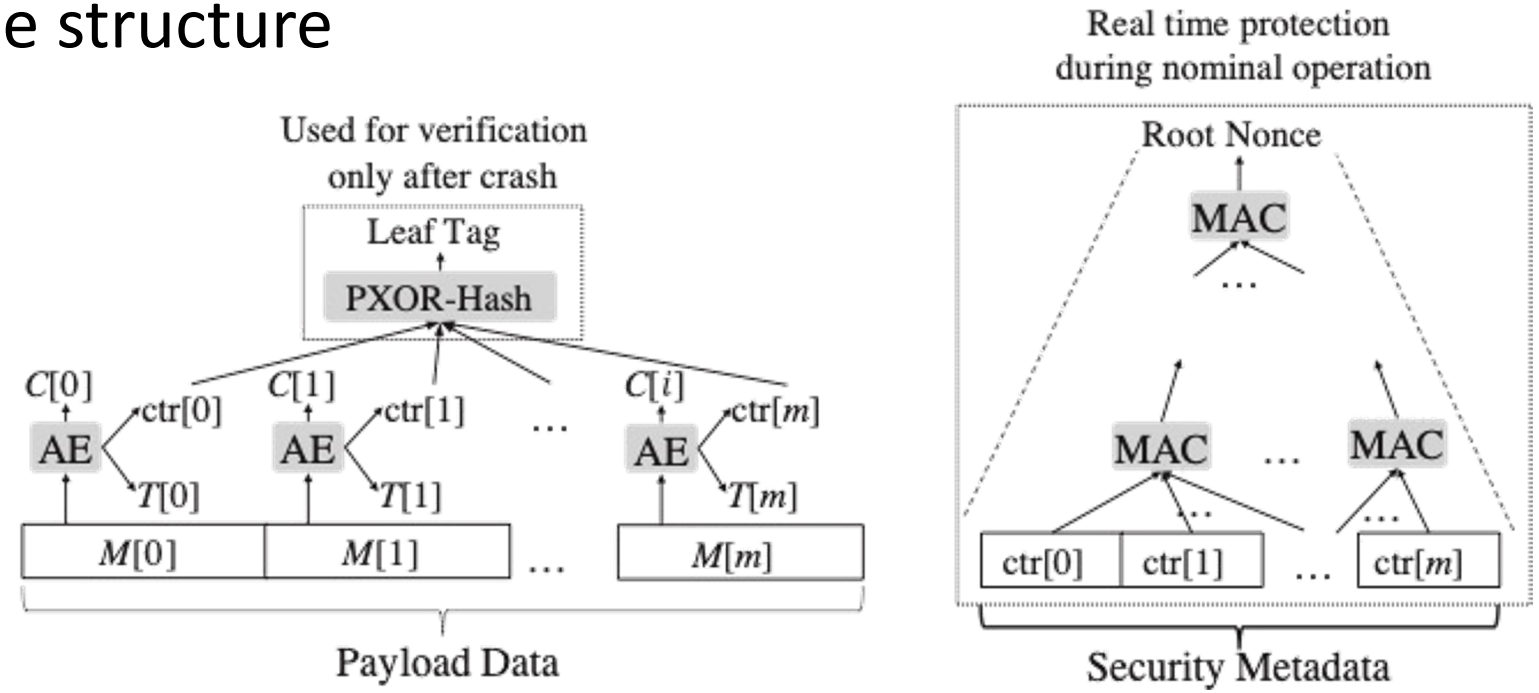
Only 120 bits for 8 nodes

Split counter (SC): Nodes share upper bits of nonce counters, where i-th nonce is given by (N_{major}‖ N[i])

[YEP+06], C. Yan et al., "Improving Cost, Performance, and Security of Memory Encryption and Authentication," *ISCA*, 2006.

Our solution: Crystalor [UHH+24]

- Achieve L4-security (i.e., recoverability of L3-secure PAT) with almost no latency overhead during operation of the CPU
- Applicable to any PATs with almost no overhead, independently of its data/nonce structure

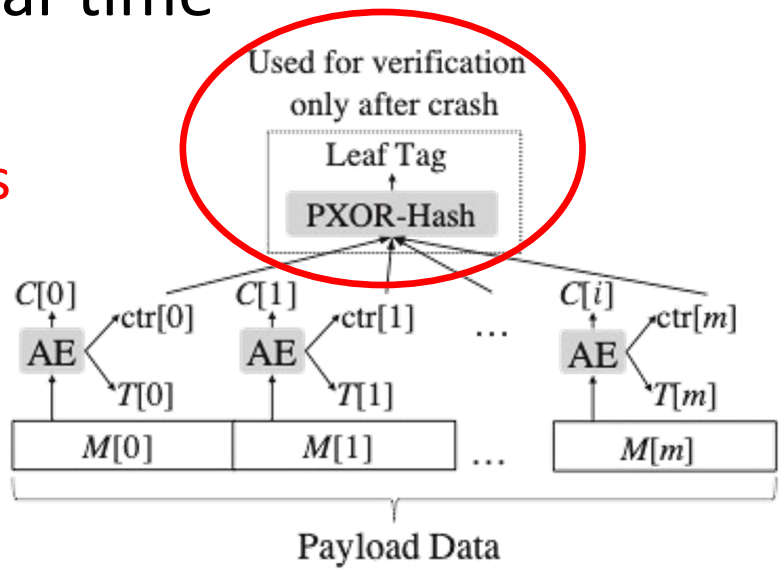


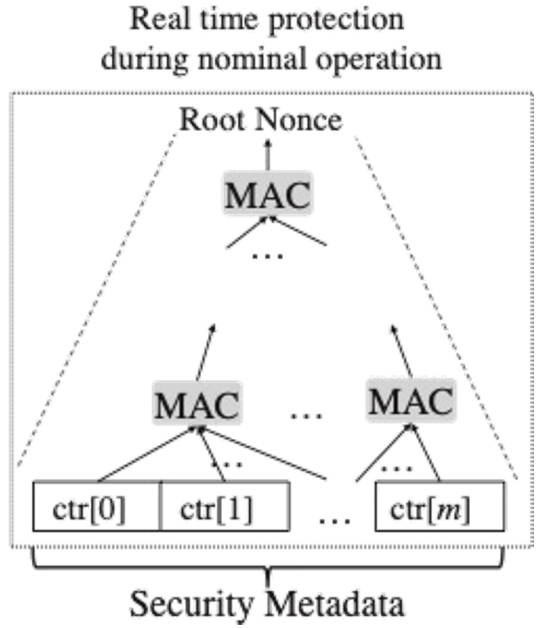
[UHH+24] R. Ueno, H. Haneda, N. Homma, A. Inoue, K. Minematsu, "Crystalor: Recoverable memory encryption mechanism with Optimized Metadata Structure," *ACM CCS*, 2024. (to appear)

Key concepts of Crystalor

- Adopt on-chip **leaf tag**, which is verified only upon crash/error detection
  - Verify only leaf nodes at reboot, and newly create intermediate nodes of tree
    - Remove necessity of path consistency during update, which is hard to realize
  - Sufficient to verify at crash/error; **verification** does not need real-time processing
- Leaf tag **update** should be real-time
  - How can we achieve it?

Consider its mandatory properties and new cryptographic primitive!

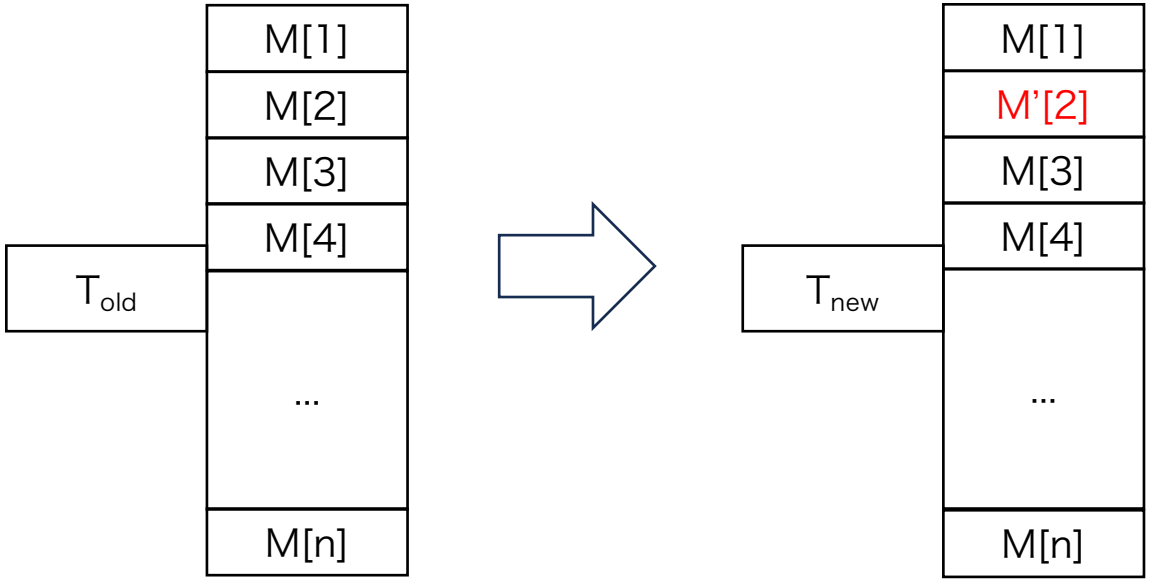




[UHH+24] R. Ueno, H. Haneda, N. Homma, A. Inoue, K. Minematsu, "Crystalor: Recoverable memory encryption mechanism with Optimized Metadata Structure," *ACM CCS*, 2024. (to appear)

Incremental cryptography [BGG+95]

- Incremental MAC can update tag with fixed number of BC calls, independently of original message length
  - If old tag, data, and nonces are available, new tag is calculated by a few BC calls
    - **It is always true for the usage of leaf tag update**



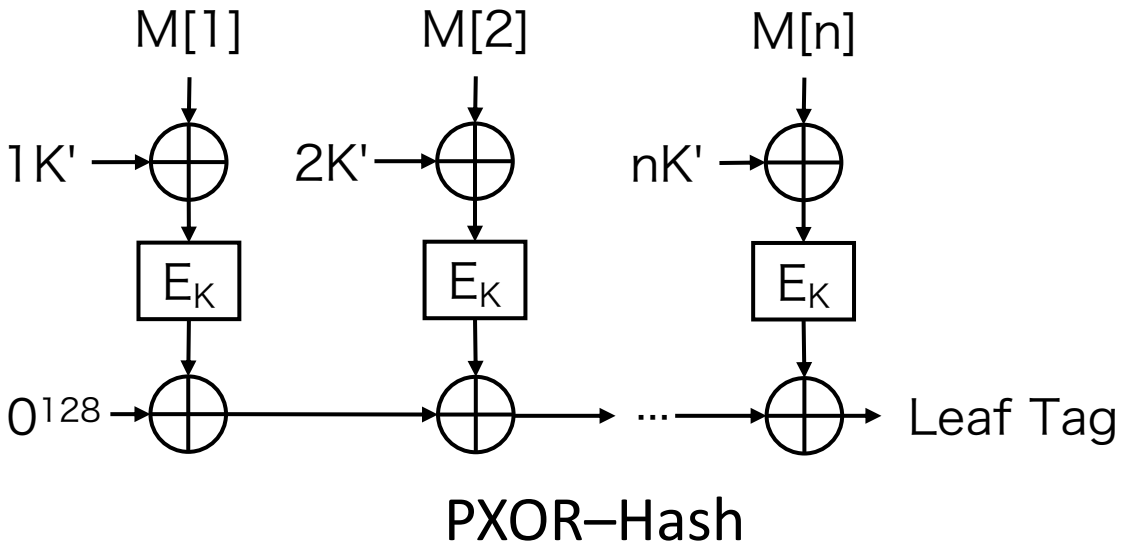
Number of BC calls

Tag generation and verification	O(n)
Non-incremental update	O(n)
Incremental update	O(1)

[BGG+95] M. Bellare, O. Goldreich, S. Goldwasser, "Incremental cryptography and application to virus protection," *STOC*, 1995.

PXOR–Hash: Specialized primitive for leaf tag

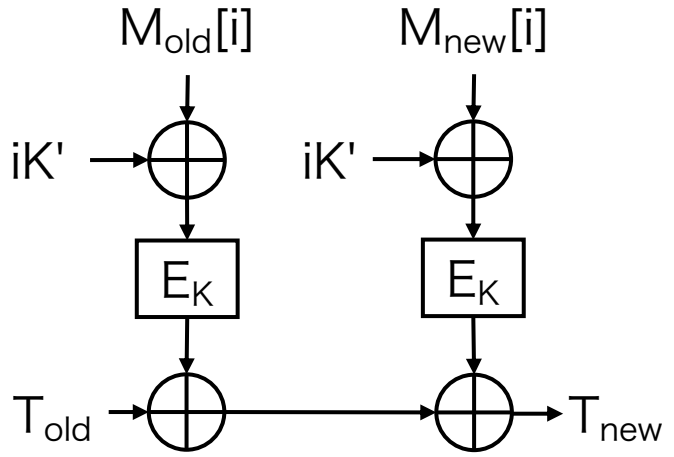
- Leaf tag is stored on-chip, meaning attacker can neither eavesdrop nor manipulate the leaf tag
  - Full-fledged MAC works, but may achieve stronger security goal than required
    - Do not need to consider adversaries seeing its output to find collision
  - Almost (XOR-)universal (A(X)U) function is sufficient for secure leaf tag
    - Realized using key-dependency more efficiently than MAC and one-way hash
- Our proposal: PXOR–Hash
  - Similar to PXOR–MAC and PMAC
  - Rete-1 (Efficient recovery)
  - Incremental update (Realtime comput.)
  - Sufficient security (Attacker cannot find collision in the scenario)



PXOR–Hash

Incremental update of PXOR–Hash

- Consider update of i-th block M[i]
- Given old tag T_{old} and old data block $M_{old}[i]$, T_{new} is computed with only two E_K calls
  - Old tag and data are always available because they must be on-chip at timing of write
- Comparison of major incremental primitives

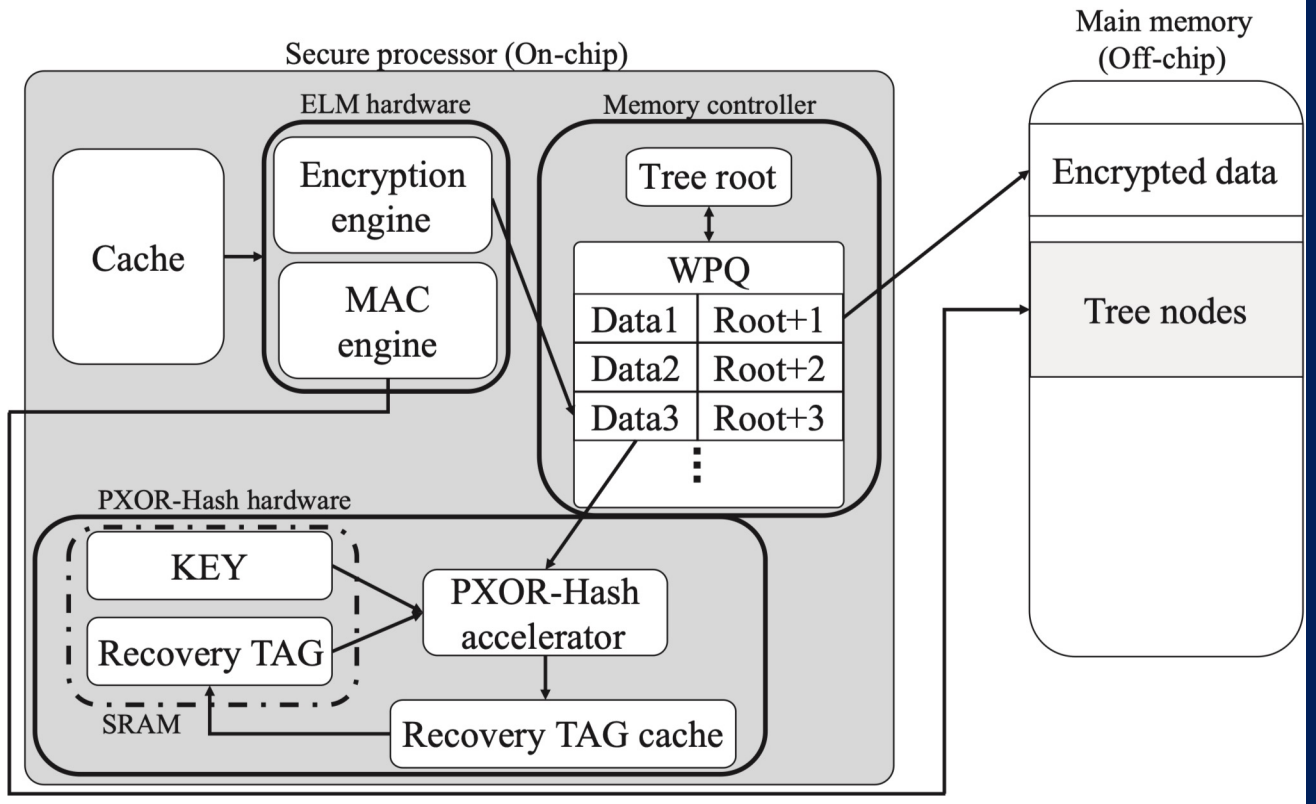


Incremental update of PXOR–Hash

	PXOR–Hash	PXOR–MAC	PMAC
Incremental update (Memory write)	# EK calls: 2 Depth: 1	# EK calls: 4 Depth: 1	# EK calls: 4 Depth: 2
Tag generation (Recovery)	# EK calls: n Depth: 1	# EK calls: n + 1 Depth: 1	# EK calls: n + 2 Depth: 2
Inverse-freeness	Yes	Yes	No (Need E_K^{-1})

Hardware architecture and implementation

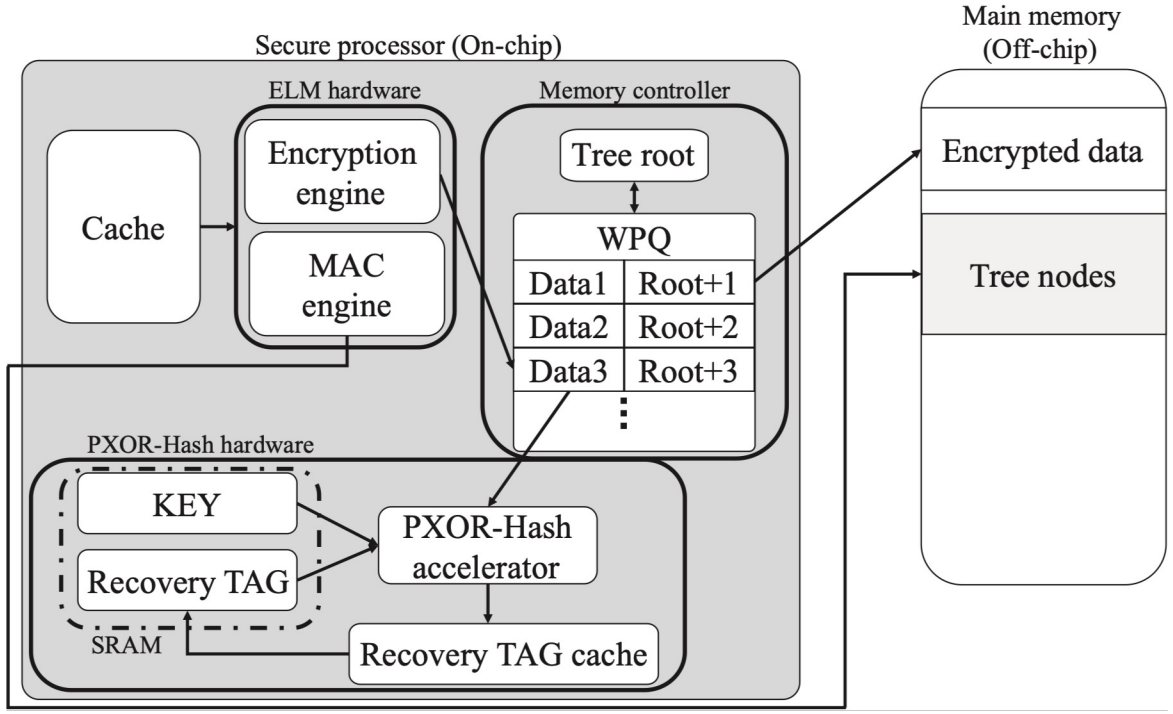
- Add PXOR–Hash hardware to existing memory encryption engine
  - Crystalor and memory encryption engine distinctly operate
  - Compatible with most existing processor architecture and optimization mechanisms
- Leaf tag is computed on-chip
  - Never expose outside
    - Secure and fast computation
  - No latency overhead as it is faster than PAT
  - Only 384-bit on-chip NVM
- Use WPQ as ADR domain
  - Used for consistent update of leaf tag and leaf nodes



Proposed hardware architecture

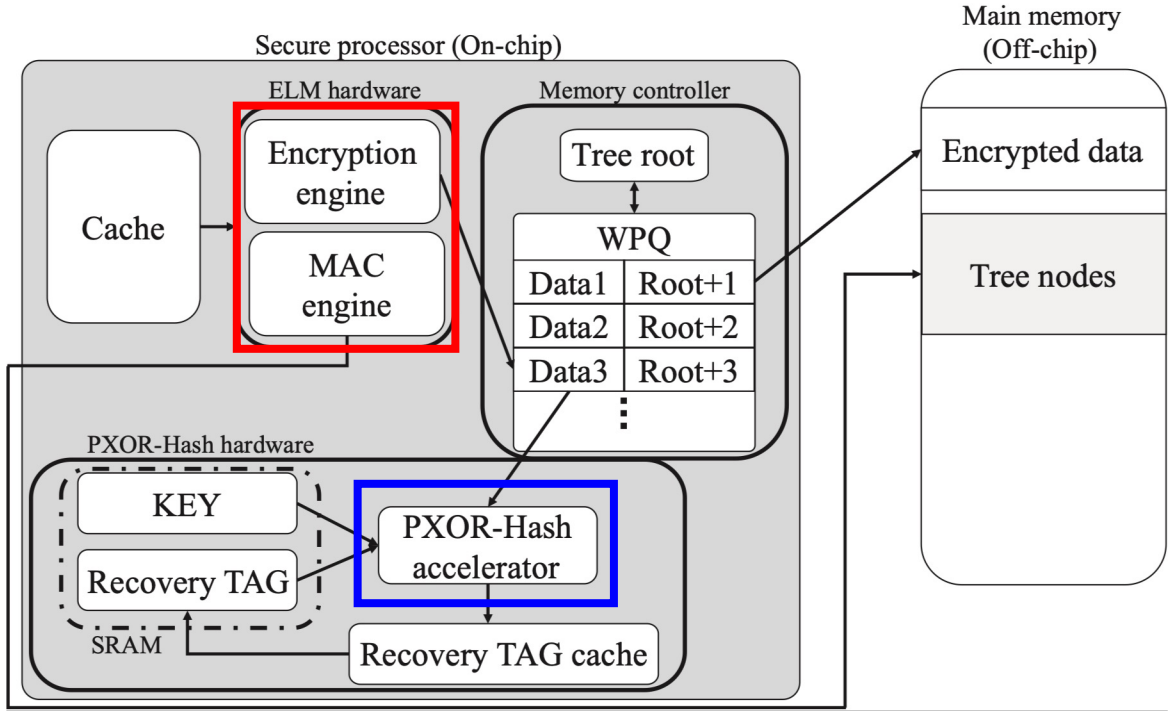
How Crystalor operates—Store operation

1. Store operation is issued



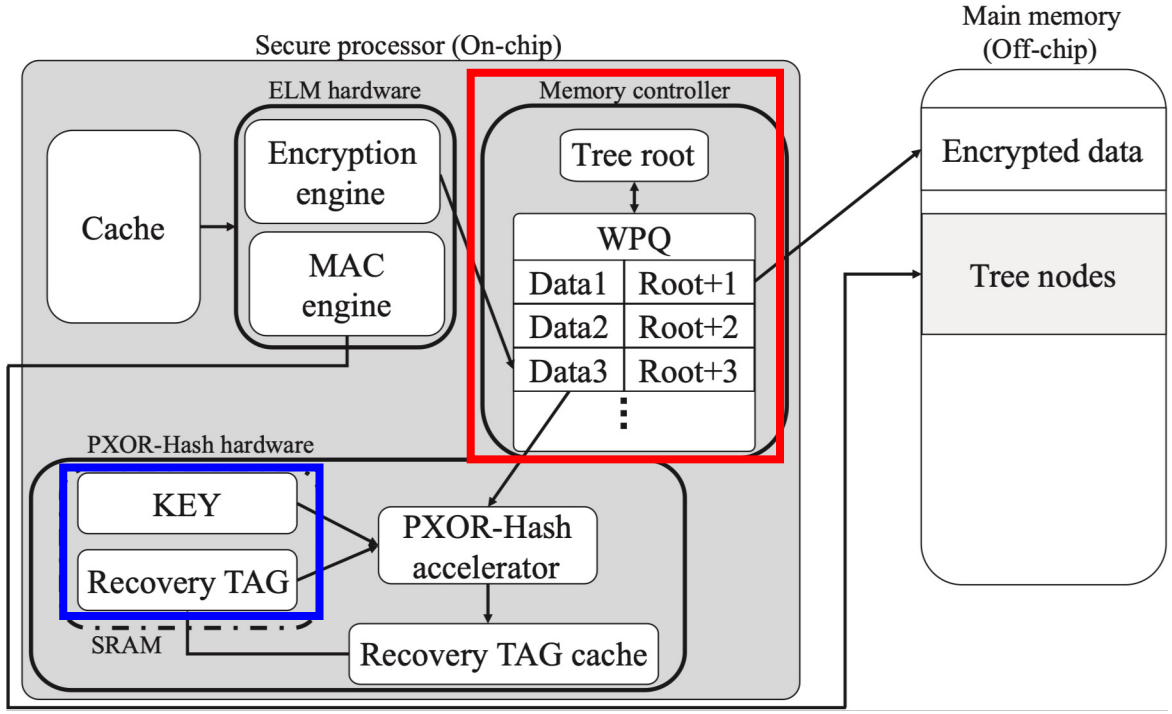
How Crystalor operates—Store operation

1. Store operation is issued
2. Parallel computation of ELM and leaf tag
   - Leaf tag input is nonce of the leaf node to be updated
   - Raise busy flag (one-bit register)
     - AE inputs (i.e., leaf node) should be preserved in NVM for recovery from crash during AE computation
     - Use ELM hardware (i.e., memory encryption engine) as ADR



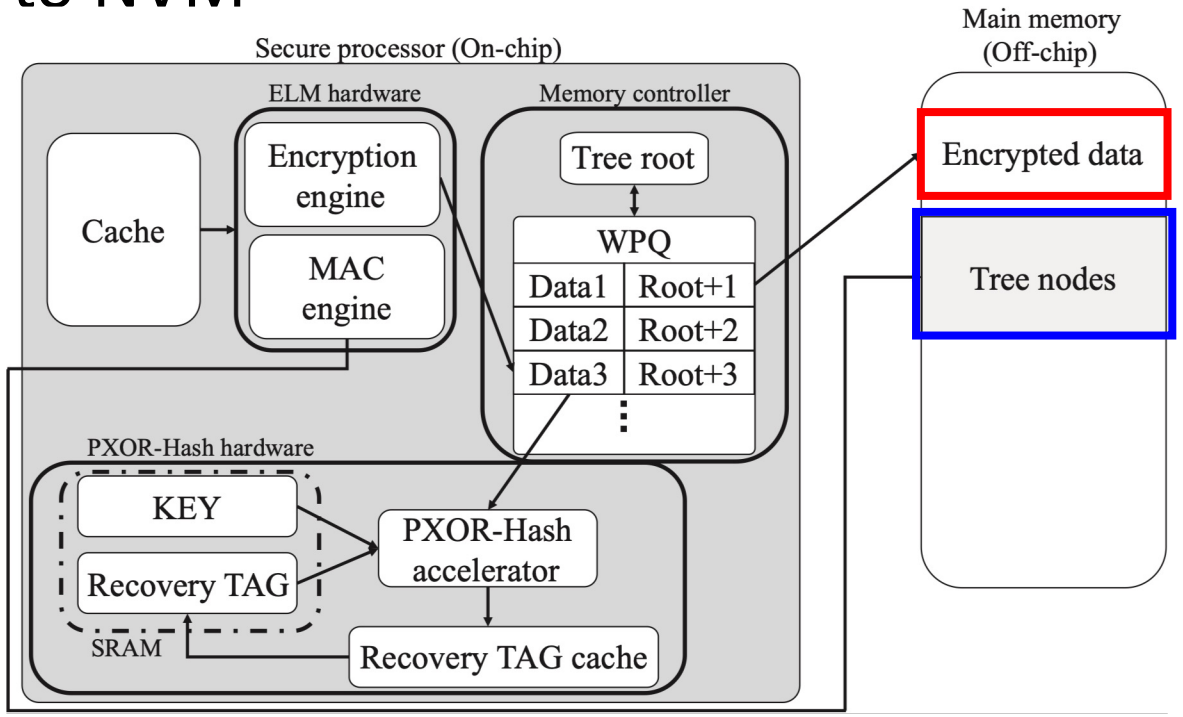
How Crystalor operates—Store operation

1. Store operation is issued
2. Parallel computation of ELM and leaf tag
3. Write encrypted leaf nodes to WPQ and update leaf tag
   - Persist leaf nodes
     - Busy flag is put down
   - Leaf tag should be consistently and simultaneously updated



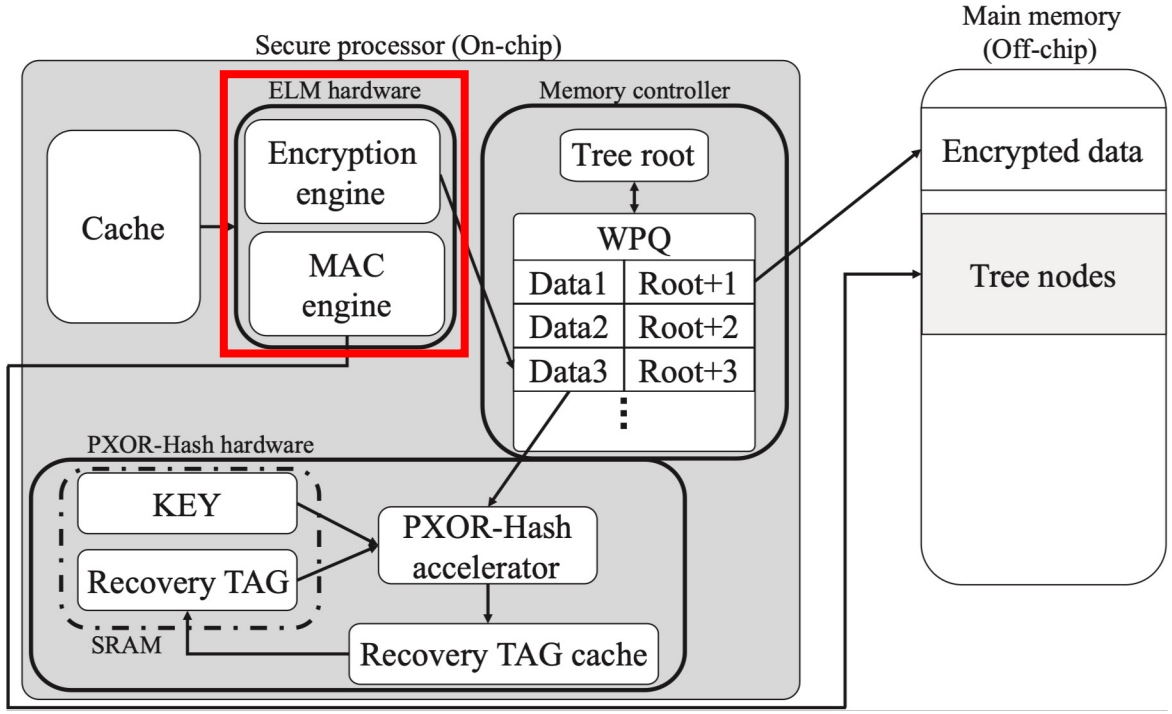
How Crystalor operates—Store operation

1. Store operation is issued
2. Parallel computation of ELM and leaf tag
3. Write encrypted leaf nodes to WPQ and update leaf tag
4. Write leaf and intermediate nodes to NVM
   - Persistency of leaf node is guaranteed because they are in WPQ
   - Intermediate nodes are directly written to NVM without WPQ
     - They do not need strong persistency because they are not used at recovery



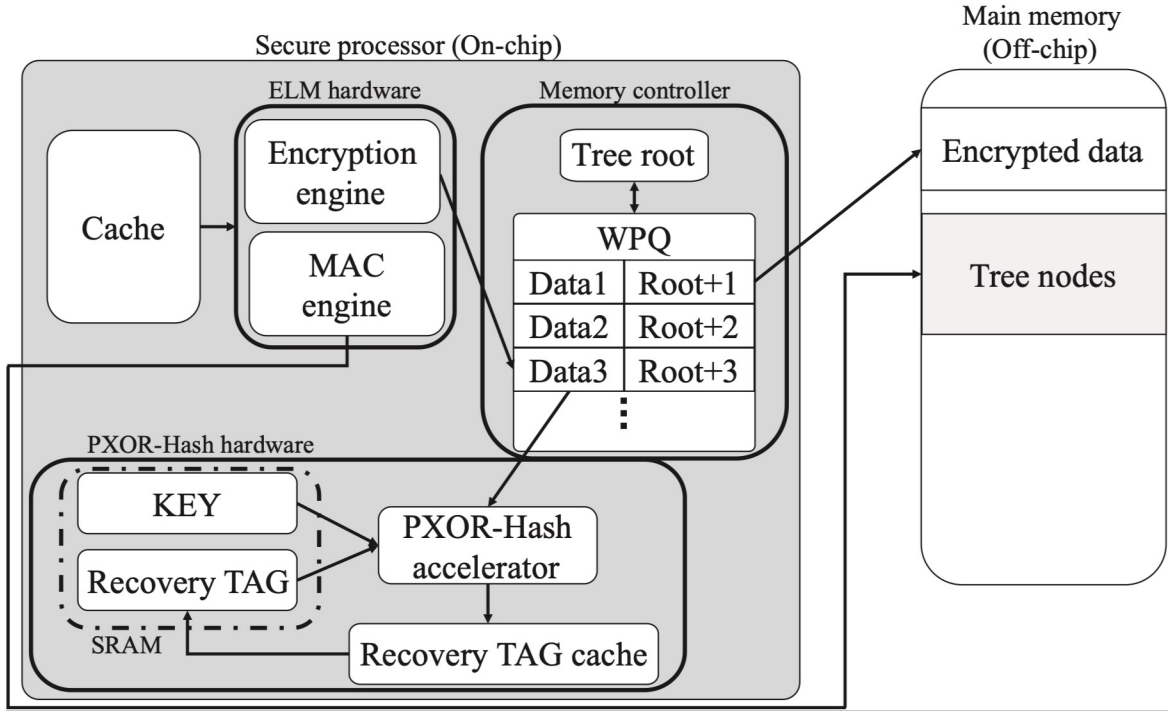
How Crystalor operates—Recovery at reboot

1. AE status check
   - If busy flag is raised, compute leaf node AE and leaf tag using data preserved in NVM, and write the result to memory through WPQ and update leaf tag
   - Otherwise, we found leaf node and leaf tag are correctly stored (nothing to do)



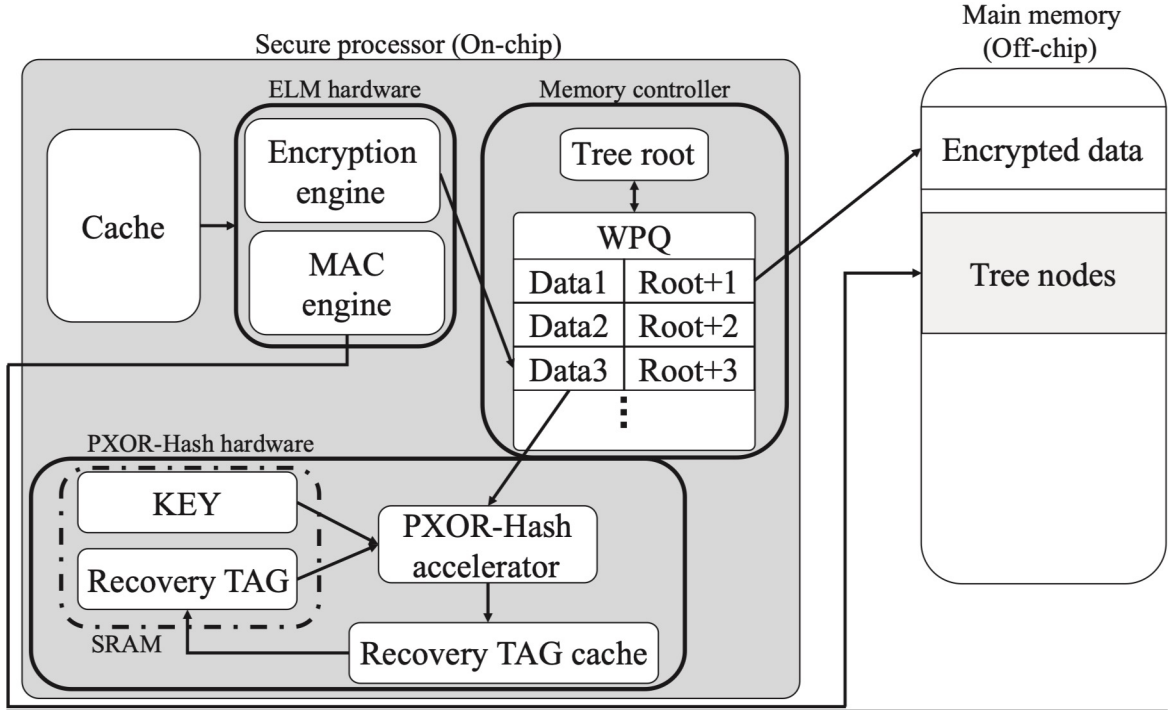
How Crystalor operates—Recovery at reboot

1. AE status check
2. Create new tree from leaf nodes in bottom up manner
   - For SC-based PAT, no way to recover intermediate node nonces before crash/error
   - Nonces of intermediate nodes are determined such that replay attack is not available
   - Use a lower bound for new nonce [Theorem 1, UHH+24]



How Crystalor operates—Recovery at reboot

1. AE status check
2. Create new tree from leaf nodes in bottom up manner
3. Leaf tag verification
   - Any manipulation of leaf node **except for replay** is detected by leaf node AE
   - Leaf tag verifies nonces are **not replayed**
   - Steps 2 and 3 should be performed in parallel to avoid manipulation during these steps



Algorithmic-level evaluation

- Evaluate latency of write and read operations for covered region size
  - Left: Covered region sizes w/o SC
  - Right: Covered region sizes w/ SC
- For example, focus on protection of 4 TB memory
  - d = 5: SC reduces latency by 62%
  - d = 7: SC reduces latency by 29%
  - SC reduces NVM overhead for metadata by 44%
  - SC's advantage directly represents Crystalor's advantage for L4 security

d: tree depth (# AES encryption engines available in parallel)

Tree arity		Latency		Covered region [Byte]					
				ELM w/o SC			ELM with SC		
b	ℓ	Update†	Verify	$d=3$	$d=5$	$d=7$	$d=3$	$d=5$	$d=7$
4	512	21	18	33 K	2 M	134 M	2 M	2 G	2 T
	1,024	25	22	66 K	4 M	268 M	4 M	4 G	4 T
	2,048	33	30	131 K	8 M	537 M	8 M	8 G	9 T
	4,096	49	46	262 K	17 M	1 G	17 M	17 G	18 T
	8,192	81	78	524 K	34 M	2 G	34 M	34 G	35 T
8	512	22	20	262 K	67 M	17 G	17 M	69 G	281 T
	1,024	25	22	524 K	134 M	34 G	34 M	137 G	563 T
	2,048	33	30	1 M	268 M	69 G	67 M	275 G	1 P
	4,096	49	46	2 M	537 M	137 G	134 M	550 G	2 P
	8,192	81	78	4 M	1 G	274 G	268 M	1 T	5 P
16	512	30	28	2 M	2 G	2 T	134 M	2 T	36 P
	1,024	30	28	4 M	4 G	4 T	268 M	4 T	72 P
	2,048	33	30	8 M	9 G	9 T	537 M	9 T	144 P
	4,096	49	46	17 M	17 G	18 T	1 G	18 T	288 P
	8,192	81	78	34 M	34 G	35 T	2 G	35 T	576 P
32	512	46	44	17 M	69 G	281 T	1 G	70 T	5 E
	1,024	46	44	34 M	137 G	563 T	2 G	141 T	9 E
	2,048	46	44	67 M	275 G	1 P	4 G	281 T	18 E
	4,096	49	46	134 M	550 G	2 P	9 G	563 T	37 E
	8,192	81	78	268 M	1 T	5 P	17 G	1 P	74 E
64	512	78	76	134 M	2 T	36 P	9 G	2 P	590 E
	1,024	78	76	268 M	4 T	72 P	17 G	5 P	1 Z
	2,048	78	76	537 M	9 T	144 P	34 G	9 P	2 Z
	4,096	78	76	1 G	18 T	288 P	69 G	18 P	5 Z
	8,192	81	78	2 G	35 T	576 P	137 G	36 P	9 Z
128	512	142	140	1 G	70 T	5 E	69 G	72 P	76 Z
	1,024	142	140	2 G	141 T	9 E	137 G	144 P	151 Z
	2,048	142	140	4 G	281 T	18 E	275 K	288 P	302 Z
	4,096	142	140	9 G	563 T	37 E	550 K	576 P	604 Z
	8,192	142	140	17 G	1 P	74 E	1 P	1 Z	1 Y

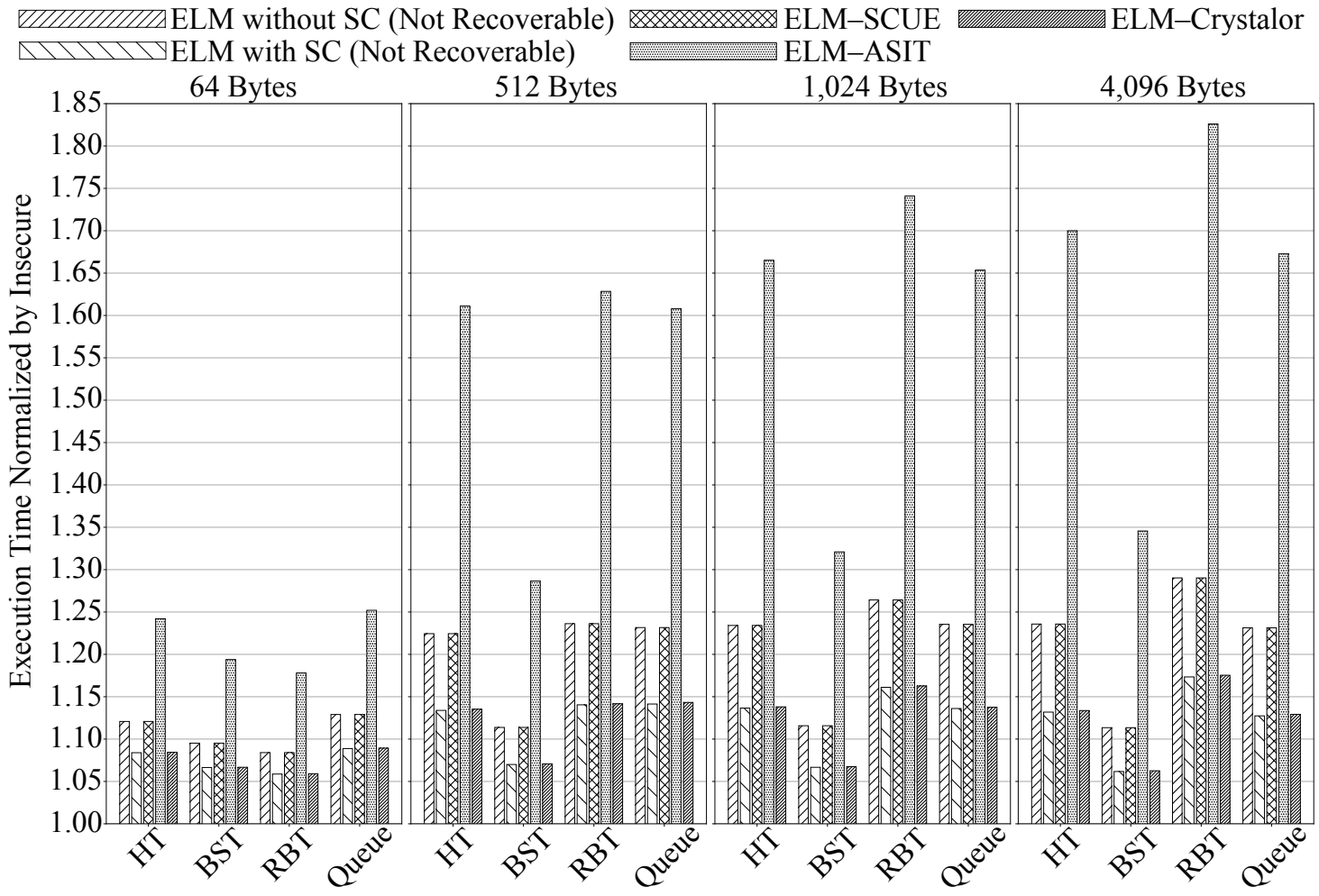
System-level simulation

- Evaluate execution times of benchmarking workloads using cycle-accurate CPU simulator gem5
  - Memory size is 4 TB
- Evaluation targets
  - Insecure (No memory encryption)
  - ELM without SC (L3, Non-recoverable)
  - ELM with SC (L3, Non-recoverable)
  - ELM–SCUE without SC (L4)
  - ELM–Anubis without SC (L4)
  - ELM–Crystalor with SC (L4)

CPU and caches	
CPU core	One core, out-of-order, 2.4 GHz
L1 instruction cache	32 KB, 8-way, 2 cycles
L1 data cache	64 KB, 8-way, 2 cycles
L2 cache	32 KB, 8-way, 2 cycles
Metadata cache	256 kB with cache line 64 byte
Memory controller and main memory (NVM)	
WPQ size	8 entries
Memory latency	Read 50 ns and Write 150 ns
Memory size (covered region)	4 TB
ELM ($d = 5$), to which SC is applied	
Update and verify latency	30 and 28 cycles
Tree parameters	$b = 16$ and $\ell = 1{,}024$
ELM ($d = 5$), to which SC is inapplicable/not applied	
Update and verify latency	78 and 76 cycles
Tree parameters	$b = 64$ and $\ell = 1{,}024$
ELM ($d = 7$), to which SC is applied	
Update and verify latency	22 and 20 cycles
Tree parameters	$b = 8$ and $\ell = 1{,}024$
ELM ($d = 7$), to which SC is inapplicable/not applied	
Update and verify latency	30 and 28 cycles
Tree parameters	$b = 16$ and $\ell = 1{,}024$

Simulation result (d = 5): Normalized execution time

- Crystalor achieved at most 11% reduction of exec. time
- Crystalor has almost same exec. time as L3-secure ELM with SC
  - PXOR–Hash computation has little impact on time
  - Because it is designed to be far faster than PAT and be computed in parallel



Concluding remarks

- Adding fault-tolerance, crash consistency, and recoverability to real-world cryptographic application is sometimes non-trivial task
  - In memory encryption, we needed to address dilemma between authenticity (security) and availability
  - Recent progress of memory encryption deeply solved this dilemma
    - Crystalor incurs almost no latency overhead for tree recovery, while it can be adopted in most existing CPU architectures and optimization mechanisms
    - Leaf nodes can be protected by ECC and memory tagging/coloring
- Memory encryption has been mainly studied in microarchitectural domain
  - Fewer studies in cryptographic and security research domains, IMO
  - Worth studying memory encryption and crash consistency problem from various perspectives, as they are interdisciplinary topics

Recovery cost estimation