Crystalor: Persistent Memory Encryption Mechanism with Optimized Metadata Structure and Fast Crash Recovery

Rei Ueno
Tohoku University, Japan
rei.ueno.a8@tohoku.ac.jp

Hiromichi Haneda
Tohoku Univesity, Japan
hiromichi.haneda.r5@dc.tohoku.ac.jp

Naofumi Homma
Tohoku University, Japan
naofumi.homma.c8@tohoku.ac.jp

Akiko Inoue
NEC, Japan
a_inoue@nec.com

Kazuhiko Minematsu
NEC, Japan
k-minematsu@nec.com

Abstract

This study presents an efficient persistent memory encryption mechanism, named Crystalor, which efficiently realizes a secure persistent/non-volatile memory based on an authentication tree with structural optimization, such as the split counter (SC). Crystalor can completely exploit the advantage of metadata compression techniques, whereas existing mechanisms are incompatible with such optimization. Meanwhile, Crystalor incurs almost no latency overhead under the nominal operation conditions for realizing the crash consistency/recoverability. We implement Crystalor with a state-of-the-art parallelizable authentication tree instance, namely ELM (IEEE TIFS 2022), and evaluate the effectiveness by both algorithmic analyses and system-level simulation in comparison with the existing state-of-the-art ones (*e.g.*, SCUE in HPCA 2023). For protecting a 4 TB memory, Crystalor requires 29–62% fewer clock cycles per memory read/write operation than SCUE owing to the compatibility with the SC. In addition, Crystalor and SCUE require 312 GB and 554 GB memory overheads for metadata, respectively, which indicates that Crystalor achieves a reduction of memory overhead by 44%. The result of the system-level simulation using the gem5 simulator indicates that Crystalor achieves a reduction of the workload execution time by up to 11.5% from SCUE. Moreover, Crystalor can offer a *lazy recovery*, which makes recovery several thousand times faster than SCUE.

1 Introduction

1.1 Background

Non-volatile memory and persistent memory, such as MRAM [18], have been deployed and attracted much attention as they can preserve data without power consumption, in contrast to volatile memories, including DRAM. Its application currently includes Internet-of-Things (IoT) devices and data centers to reduce power/energy consumption [19,34]. Additionally, in some recent systems, non-volatile and persistent memories have been deployed as main memory in addition to/instead of DRAM for higher performance, lower power consumption, and larger capacity, such as NVDIMM and Intel Optane Persistent Memory [2]. In this study, we refer to non-volatile and persistent memories as "NVM" collectively.

In deploying NVM to practical systems, there are two challenging issues: security and crash consistency. The security means that NVM data should be protected against attacks (*i.e.*, eavesdropping and manipulation/forgery of confidential data in the NVM) as well as DRAM [26]. Such attacks are more serious and realistic for NVM than DRAM due to the non-volatility. In general, *memory encryption* has realized the (NVM) security based on symmetric cryptography [24], such as encryption and message authentication code (MAC). For example, the Optane module is equipped with a 256-bit AES–XTS engine for data privacy. Here, splicing and replay attacks, which are the data moves in the spatial and time domains, respectively (*i.e.*, copy-and-paste of data in another address and at past timing, respectively), are sophisticated manipulation/forgery. These attacks cannot be protected by the simple use of encryption and MAC, as the manipulated data had been originally valid.

In the memory encryption, Merkle tree [48, 49] and parallelizable authentication tree (PAT) [27] have been developed for protection against extensive attacks with a realistic computational latency. For example, Intel SGX integrity tree (SIT) [14, 24, 25] is a popular PAT instance. Each leaf node consists of encrypted data, a counter value (*i.e.*, nonce), and a verification tag, whereas each intermediate/root node consists of a nonce and a tag to verify its child node as security metadata. When reading (resp. storing) data corresponding to a leaf node, all nodes on the path from the leaf to root nodes are verified (resp. updated). Such a tree structure enables real-time processing of verification and update because its path verification is far faster than the MAC computation for the whole NVM. In addition, the root node, which consists of only several hundred bits, is stored on-chip, as the attacker supposedly cannot manipulate on-chip data. Thus, the root node acts as a root of trust to preclude replay attacks.

Crash consistency means a guarantee that NVM data is

not broken after a crash (*e.g.*, sudden power-off/blackout) occurs [45, 82]. A write pending queue (WPQ) is employed in CPUs as an asynchronous DRAM refresh (ADR) domain to persist data to be stored in NVM. For secure NVM based on Merkle tree or PAT, however, it is mandatory to guarantee the consistency of all data, including metadata of intermediate nodes. If the tag verification (on intermediate nodes) fails, the related data (leaf node) becomes unavailable because it may have been manipulated. However, it is non-trivial to guarantee crash consistency of the whole tree because a path is to be updated serially in practice (although the computation may be parallelized [27]). This indicates that there exists a moment when a path is inconsistent (*i.e.*, some nodes are updated while others are not yet). Thus, it is challenging to efficiently guarantee consistency against sudden crashes whenever the system operates.

PAT vs. Merkle tree. In PAT, the MAC tags on a path can be updated in parallel. PAT enables algorithmically lower latency than Merkle tree since the path update of Merkel tree is not parallelizable. In contrast, Merkle tree realizes crash consistency more easily than PAT [30, 82]. In this study, we develop an efficient crash consistency mechanism of PAT, which is promising to improve persistent memory encryption.

State-of-the-art. In HPCA 2023, Huang and Hua presented a crash recovery mechanism for PAT, named *shortcut update (SCUE)* [30]. SCUE exploits a simple property of naïve PAT metadata; that is, a parent node counter is always equal to the sum of child node counters. SCUE requires almost no overhead when applied to a PAT-based secure NVM, and is currently most efficient to realize a secure recoverable NVM.

Remaining problem. SCUE is based on a property of counters in a naïve PAT *without optimization*. Meanwhile, several PAT structural optimizations using advanced counter mechanisms to compress security metadata have been presented, which significantly reduce the overhead incurred by memory encryption. *Split Counter (SC)* is the representative and most typical advanced counter mechanism [74], followed by, *e.g.*, *Vault* [65] and *Morphable Counters* [62]. SCUE has a major and critical drawback: they do not work together with such structural optimizations using advanced counter mechanisms that do not preserve the aforementioned simple property of counters. Other major crash consistency mechanisms are also incompatible with the optimizations [82]. Indeed, *combination of crash recovery and such optimization techniques has rarely been studied and exploited thus far, and there has been a mismatch between them*. Owing to the advantage of optimization techniques, it is important to develop a persistent memory encryption mechanism for PAT compatible with them for the deployment of secure NVM.

1.2 Our contributions

We propose a persistent memory encryption mechanism, named *Crystalor*, which stands for *CRYptographically Secure Tree-based Authentication with Leaf-Only Recovery*. Crystalor achieves crash recoverability with almost no overhead in terms of latency, whereas it is compatible with structural optimizations, such as SC. Our ideas are twofold. First, Crystalor does *not* guarantee the consistency/recovery of the whole tree but guarantees the integrity and recovery of only leaf nodes (*i.e.*, payload data) using a distinct verification tag (leaf tag) against a crash. Second, after a crash, Crystalor relinquishes the tree except for leaf nodes and creates a new tree from the verified leaf nodes with resilience against replay attacks.

In Section 3, we present our key observations: (1) an almost universal (AU) hash function [13] is sufficient for a secure leaf tag verification, and (2) it is possible to build an efficient AU hash function with two properties, namely rate-1 (*i.e.*, one block cipher calls to process one input block) and incrementality. Based on this observation, we develop a (computational) AU hash function named PXOR-Hash, which is tailor-made for our purpose. PXOR-Hash is designed similarly to PXOR-MAC, which is used in state-of-the-art PAT named ELM [32]; PXOR-Hash is designed with a philosophy similar to PXOR-MAC; however, PXOR-Hash achieves another security goal/level and has a construction different from PXOR-MAC, which yields highly efficient implementation[1].

The performance advantage of Crystalor is extensively evaluated by algorithmic analyses and a system-level simulation. Our results reveal that, for protecting a 4 TB NVM as an example, Crystalor achieves 29–62% latency reduction per memory read/write in the algorithmic level and a 44% reduction of NVM overhead compared to SCUE owing to the compatibility with SC. The system-level simulation using the gem5 simulator [9] shows that Crystalor achieves at most 11.5% lower latency than SCUE. In addition, Crystalor can adopt a *lazy recovery* while SCUE cannot, which makes recovery faster than SCUE by several thousand times.

1.3 Related works

Crash consistency of secure NVM. In [79], Mao *et al.* presented *Osiris*, which recovers the tree only from counter values by exploiting error-correcting code (ECC) bits equipped with NVM. In [76], Yang *et al.* presented *cc-NVM*, which caches flushed counter values in WPQ and employs a MAC in contrast to Osiris. In [6], Awad *et al.* presented *Triad-NVM* for the recovery of Merkle tree, which reconstructs the tree from flushed nodes. In [82], Zubair and Awad presented *Anubis*. It utilizes a *shadow table* stored in NVM, which contains information on cached nodes and identifies and recovers non-updated nodes. In [3], Alwadi *et al.* presented *Phoenix* that

[1] Note that our major contribution includes presenting a new persistent memory encryption mechanism with how to use cryptographic primitives to realize a secure and recoverable NVM and its concrete construction, rather than cryptographic aspects (*e.g.*, development of new proof technique or refreshingly new cryptographic primitive). We developed Crystalor with a simple construction for leveraging the existing cryptographic theories and for the compatibility of existing architectures and optimization mechanisms.

combines Osiris and Anubis. In [75], Yang *et al.* presented *ShieldNVM*, which introduces an epoch-based mechanism to aggressively cache the metadata with the consistency preserved. In [29], Huang and Hua presented *STAR* to achieve a reduction of write overhead and fast recovery, which exploits SIT lazy scheme and instant persistency for modifications in the cache. In [83], Zubair *et al.* investigated the error sensitivity of metadata in Merkle tree and presented *Soteria* to tolerate the errors by its lazy duplication.

Advanced mechanisms for counter and tree structures. The SC is the pioneering advanced counter mechanism [74], which compresses and optimizes the tree/metadata structure by splitting nonce counters into major and minor counters. In [65], Taassori *et al.* presented *Vault*, which adjusts the tree arity to reduce the frequency of counter overflow and improve the capacity of the covered region. In [62], Saileshwar *et al.* presented *Morphable counters*, which dynamically/adaptively determines the structure of major and minor counters in SC, and enables to cache more counters in a line. In [81], Zhou *et al.* presented *Lelantus*, which improves the counter-mode AES-based secure NVM by exploiting fine-granularity copy-and-write operations. In [20], Freij *et al.* presented *Bonsai Merkle Forest*, which splits Merkle tree into subtrees.

1.4 Paper organization

Section 2 introduces the persistent memory encryption and secure NVM. Section 3 presents the proposed crash recovery mechanism Crystalor, and explains its hardware architecture and operation. Section 4 demonstrates the algorithm-level evaluation regarding SC and a system-level simulation using the gem5 simulator. Finally, Section 5 concludes this paper.

2 Preliminaries

2.1 System and threat models of secure NVM

Figure 1 shows a system model with secure NVM, which follows many existing studies on secure NVM. We here omitted CPU core(s) and a memory controller to focus on our interest. The model separates the memory system into two areas: the on-chip trusted area and the off-chip untrusted area. On-chip data are assumed to be secure and trustable; that is, any attacker can *neither* eavesdrop nor manipulate on-chip data. In contrast, he/she can perform arbitrary eavesdropping/manipulation of off-chip data. The goal of NVM security is to realize the confidentiality and integrity of NVM data. Thus far, block cipher (*e.g.*, AES) has been used for confidentiality, while MAC (*e.g.*, HMAC) is for integrity (*i.e.*, the detection of data manipulation/forgery) [21]. The on-chip area contains cache(s), memory protection hardware, SRAM, and a WPQ. The SRAM stores the secret key and root nonce of PAT. The memory protection hardware performs cryptographic computation including AES and MAC. The CPU systems employ a WPQ as an ADR domain to buffer data from the cache to NVM until it is written, which guarantees the data persistency at the time of cache data replacement/flush. This threat model is compatible with many existing studies on memory encryption [21, 24, 30], with a focus on our context. See [24] for the formalization.

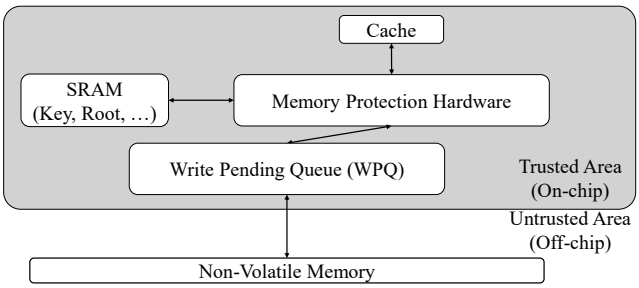


Figure 1: System model with secure NVM (CPU core(s) and memory controller are omitted).

Remark 2.1 (Side-channel attacks). We do not consider side-channel attacks as in previous studies. For example, power, EM, timing, and cache analyses on cryptographic primitives may steal the secret key [37, 38, 66, 67]. Micro-architectural attacks represented by Flush+Reload and Prime+Probe [44, 77], which are exploited by Spectre and Meltdown [36, 43], may also be a potential threat on the confidentiality of payload data. Side-channel attacks are attempts to eavesdrop and manipulate *on-chip* data. Hence, they are outside our scope; we trust the confidentiality and integrity of on-chip data. Side-channel attacks should be countered by different means, such as masking [23, 47] and cache randomization [12, 57, 58, 72] (although there is an authentication tree named MEAS that claims a power/EM side-channel security [69]).

Remark 2.2 (Relation to TEE). Some trusted execution environment (TEE) mechanisms, such as Intel Software Guard eXtension (SGX) [14] and AMD Secure Encrypted Virtualization (SEV) [1] support memory (DRAM) encryption for the resource isolation, remote attestation, *etc*. A goal of TEE memory encryption is to realize trust and privacy against *on-chip* threats (*e.g.*, malicious or compromised privileged user, OS, BIOS, cloud vendor, VMM, and spy process), while its focus may not include side-channel attacks as well [11, 40–42, 51–53, 70, 73]. TEE memory encryption may not support (full) authenticity nor require crash consistency. The persistent memory encryption provides stronger security (*i.e.*, full authenticity and crash consistency in addition to privacy), as it considers physical access to the NVM. Thus, it would contribute to TEE memory encryption. In fact, Intel SGX protects the memory using SIT, which is a well-known PAT for persistent memory encryption and secure NVM.

2.2 Symmetric cryptography for persistent memory encryption

Existing works on persistent memory encryption/secure NVM (*e.g.*, [16]) frequently employed AES encryption. For confidentiality, the counter-mode AES has been utilized due to its parallelizability and random accessibility. A nonce is also used for the encryption as security metadata, which is composed of memory address and counter. For integrity, a MAC is commonly employed in PAT. Note here that replay attacks cannot be prevented by simple use of MAC, because they manipulate data using valid data, nonce, and tag, which motivates the usage of PAT for persistent memory encryption.

Recently, for encrypting and verifying payload data, some state-of-the-art PATs, such as ELM [32], utilize an authenticated encryption (AE) [59] for leaf nodes instead of a composition of the counter-mode AES encryption and a MAC [8]. For efficient memory encryption, AE should have two desirable properties: block-level parallelizability[2] and rate-1. A parallelizable AE can encrypt several blocks in parallel, which allows for low-latency implementation using a multi-core or pipelining. Rate-1 is a property related to the number of AES calls to complete the encryption and tag computation. A rate-1 parallelizable AE (*e.g.*, [61]) has a latency of almost half of a composition of the counter-mode AES encryption and a MAC. The usage of AE significantly improves the performance of persistent memory encryption, in which payload data essentially requires both confidentiality and integrity.

Parallelizability and rate-1 are also desirable for MAC to achieve the integrity of intermediate nodes. Furthermore, some MACs have another desirable property called incrementality [7]. When a data block is changed, an incremental MAC can update the tag with $O(1)$ calls of the underlying cryptographic primitive using the old data block and tag, whereas typical non-incremental MAC, such as CMAC [17], requires $O(m)$ primitive calls, where m is the number of input blocks. The tag of incremental MAC can be updated only from an old tag, old data block, and new data block, as well as old and new nonces. In a path update of secure NVM, the number of blocks updated in MAC computation is usually one; hence, usage of incremental MAC significantly improves the performance of memory encryption.

2.3 Parallelizable authentication tree (PAT)

PAT has been employed for memory encryption to realize (1) real-time processing and (2) protection against replay attacks with a minimal overhead of on-chip memory.

Figure 2 illustrates an overview of PAT with an arity of two as an example. PAT encrypts the leaf node using an AE and verifies intermediate nodes using a nonce-based MAC. The i-th intermediate node of PAT (including the root node) consists of $(N[i], T[i])$, where $N[i]$ and $T[i]$ are its nonce and tag, respectively. The i-th leaf node of PAT consists of $(N[i], T[i], C[i])$, where $C[i]$ is the ciphertext of payload data. The nonce of each node is given by a concatenation of its address $\mathsf{addr}[i]$ and counter $\mathsf{ctr}[i]$ as $N[i] = \mathsf{addr}[i] \,\|\, \mathsf{ctr}[i]$, where $\cdot \,\|\, \cdot$ denotes the bit concatenation. Here, the tag of intermediate nodes is computed from the secret key, its nonce, and child node counters to be verified (*e.g.*, $\mathsf{ctr}[i-2] \,\|\, \mathsf{ctr}[i-1]$ in Figure 2). That is, the tag of an intermediate node verifies the integrity of its child node counters as the MAC input. For a leaf node, the ciphertext and its tag are computed from the secret key, its nonce, and plaintext (*i.e.*, payload data) in storing data, whereas plaintext and its verification tag are computed from the secret key, its nonce, and ciphertext in loading data. In both verification and update, the tag of each node in a path is computed in parallel as its computation does not depend on the computation results of any lower-level nodes, indicating the node-level parallelizability of PAT.

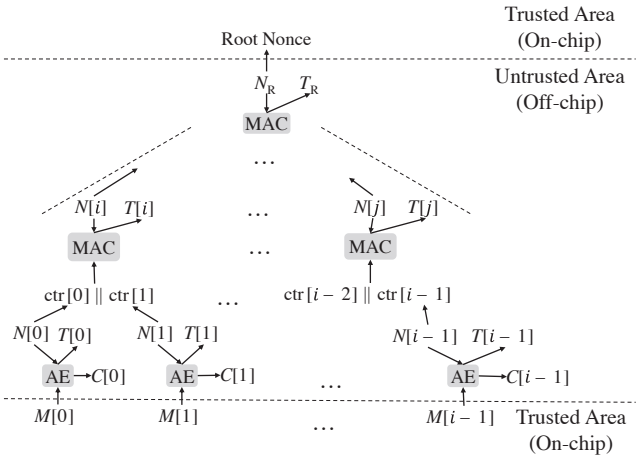


Figure 2: Overview of binary PAT. $M[i]$, $C[i]$, $N[i]$, and $T[i]$ denote plaintext (*i.e.*, payload data), ciphertext, nonce, and tag of i-th node, respectively, where $N[i]$ consists of address $\mathsf{addr}[i]$ and counter $\mathsf{ctr}[i]$. N_R and T_R are root nonce and tag, respectively. Leaf node is defined as $(N[i], T[i], C[i])$, whereas other nodes are defined as $(N[i], T[i], C[i])$.

Recall that the nonce of the i-th node is defined as $N[i] = \mathsf{addr}[i] \,\|\, \mathsf{ctr}[i]$, in which the counter is incremented when updating the node. Here, $\mathsf{addr}[i]$ is unique to the node, and $\mathsf{ctr}[i]$ does not take the same value until an overflow occurs, which guarantees no nonce collision over the whole tree. Note that $\mathsf{addr}[i]$ does not need storing in on-chip/NVM because it is implicitly determined from its physical address [14, 32].

ELM. ELM is a state-of-the-art PAT proposed by Inoue *et al.* [32]. ELM is optimized for low latency and scalability to large memory. For this purpose, Inoue *et al.* introduced an AE and an incremental MAC named Flat-OCB and PXOR-MAC, respectively, which are optimized for low latency of PAT verification/update. Moreover, ELM unifies some computations

[2] Block-level parallelizability is a property of mode and MAC, whereas node-level parallelizability is of the tree.

shareable with Flat-OCB and PXOR-MAC among the whole tree. ELM has a lower latency and less memory overhead than Intel SGX integrity tree (SIT) [14], while both schemes use AES-128 and have equivalent provable security reduction to AES. Notably, in the research field of secure and persistent memory, previous studies (*e.g.*, [21, 30]) employed a classical HMAC, which has a larger latency due to serial structure and does not have incrementality. The MAC of ELM (and SIT) is far faster than the HMAC.

2.4 Shortcut update (SCUE)

SCUE is a state-of-the-art persistent memory encryption mechanism with PAT presented by Huang and Hua in HPCA 2023 [30]. In SCUE, the counter of the nonce in PAT is *incremented by one* whenever the node is updated and never decreases under nominal operation (without any reset nor overflow). Its security against replay relies on the fact that a replay attacker can decrease a counter of a node by replacing the nonce and tag in the past but cannot increase it.

The proposals of SCUE include (i) efficient integrity verification of leaf nodes and (ii) how to reconstruct intermediate nodes from the leaf nodes. The basic ideas behind SCUE are that, unless there is any manipulation, (i) the root counter is always equivalent to the sum of all leaf counters, and (ii) a parent node counter is always equivalent to the sum of its child node counters. These facts are apparent because the counter represents the number of updates of the node. After a crash, the integrity of leaf nodes is verified first using AE/MAC with the tag and nonce stored in NVM. Assuming the security of AE/MAC, the attacker cannot perform any forgery except for replay. Then, to detect a replay of leaf nodes, the SCUE checks the equivalence between the root counter and the sum of leaf counters. Here, the root counter is manipulation-free because it is on-chip. In addition, a replay decreases a counter but cannot increase it, indicating that the sum of leaf counters is to be fewer than the root counter if replayed.

After the verification, SCUE recovers the intermediate nodes before the crash in a bottom-up manner. It determines each parent node counter value as sum of its child node counters because they are always equivalent unless manipulation.

2.5 Split Counter (SC)

SC is the foremost advanced counter mechanism for PAT structural optimization [74], which compresses the tree size (*i.e.*, suppresses the NVM overhead to store metadata) [5, 55, 56]. The tree size compression contributes to low latency because the number of input blocks to MAC/AE for verification/update of the tree becomes smaller.

Figure 3 illustrates an overview of an SC-based PAT with an arity of two. An SC-based tree splits a counter into major and minor counters. In computing AE for a leaf node, the nonce is given by a concatenation of its address, its major counter,

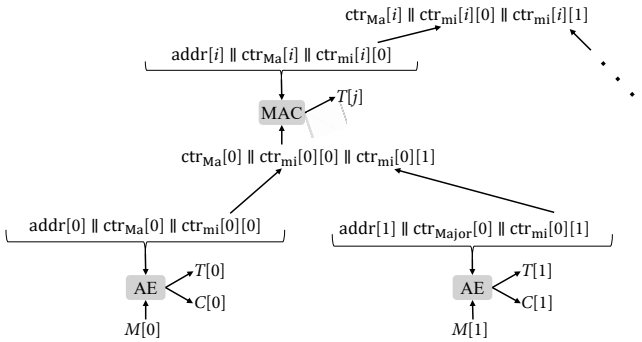


Figure 3: Example of SC-based binary PAT, where $\mathsf{ctr}_{\mathsf{Ma}}[i]$ denotes i-th major counter, and $\mathsf{ctr}_{\mathsf{mi}}[i][j]$ denotes j-th minor counter sharing $\mathsf{ctr}_{\mathsf{Ma}}[i]$. Each node has a nonce consisting of its address, major counter, and minor counter, whereas the MAC input of a parent node consists of a major counter and all minor counters of child nodes. All major and minor counters and tags are stored in NVM as security metadata in addition to ciphertext, while the root (major) counter is stored on-chip.

and its minor counter (*e.g.*, $\mathsf{addr}[0] \parallel \mathsf{ctr}_{\mathsf{Ma}}[0] \parallel \mathsf{ctr}_{\mathsf{mi}}[0][0]$ in Figure 3) and the input (*i.e.*, data to be encrypted/decrypted and verified) is payload data (*e.g.*, $M[0]$). In computing MAC of an intermediate or root node, the nonce is given by its address, its own major counter, and its own minor counter (*e.g.*, $\mathsf{addr}[i] \parallel \mathsf{ctr}_{\mathsf{Ma}}[i] \parallel \mathsf{ctr}_{\mathsf{mi}}[i][0]$) and the input (*i.e.*, data to be verified) is a concatenation of major counter(s) and all corresponding minor counters of its child node (*e.g.*, $\mathsf{ctr}_{\mathsf{Ma}}[0] \parallel \mathsf{ctr}_{\mathsf{mi}}[0][0] \parallel \mathsf{ctr}_{\mathsf{mi}}[0][1]$). Here, if a minor counter overflows in an update of node, then all minor counters that share a major counter are reset to zero, and the major counter is incremented. Thus, the SC significantly reduces the total bit length of counters, maintaining the uniqueness of nonce.

Let l_{ctr} be the bit length of the counter without SC. Let l_{Ma} and l_{mi} be those of the major and minor counters of an SC-based PAT, respectively. Typically, l_{ctr}, l_{Ma}, and l_{mi} are 64, 56, and 8, respectively [31, 74]. If k nodes share a major counter, the SC reduces the counter size from kl_{ctr} to $l_{\mathsf{Ma}} + kl_{\mathsf{mi}}$.

3 Proposed mechanism: Crystalor

3.1 Basic concept of Crystalor

Figure 4 illustrates the proposed crash recovery mechanism for PAT, named Crystalor. Crystalor distinctly provides crash recoverability and security against crashes, while PAT solely provides confidentiality and integrity *under nominal operation without a crash*. The basic ideas behind Crystalor are to use a distinct *leaf tag*, which verifies the leaf node integrity *only after a crash* but can be updated with almost no overhead under nominal operation, and to construct a new tree with a proven resilience against replay after the leaf tag verification.

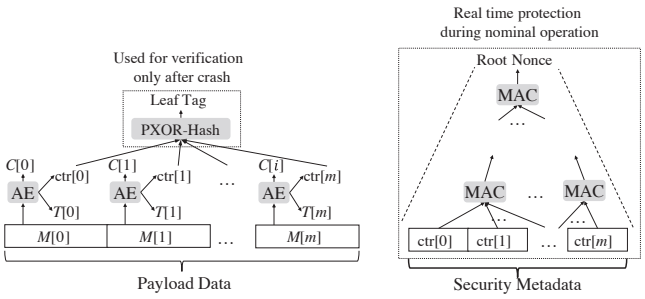


Figure 4: Secure NVM based on PAT and Crystalor.

If we can verify the leaf nodes (*i.e.*, payload data) regarding replay attacks without intermediate node consistency, the intermediate node is no longer required. Thus, we give up the whole tree consistency and relinquish the intermediate and root nodes. Crystalor realizes the leaf node verification using the leaf tag stored on-chip. The leaf tag verification is only performed after a crash, while PAT provides integrity under nominal operation. This indicates that the leaf tag verification does not need real-time processing[3]. In contrast, the tag update requires real-time processing because it should be performed whenever storing data. Thus, we present an incremental universal hash [13] named PXOR-Hash, which is tailor-made for an efficient and optimal realization of such leaf tag. PXOR-Hash is designed similarly to PXOR-MAC and PMAC; however, PXOR-Hash and PXOR-MAC/PMAC achieve different security goals/levels the difference in their contexts (see Section 3.2), which enables PXOR-Hash to improve efficiency and latency compared to PXOR-MAC/PMAC. Importantly, PXOR-Hash verifies any data regardless of its structure, which enables the integrity verification of PAT with structural optimizations (in contrast to SCUE).

The proposal of Crystalor includes how to rebuild the whole tree (*i.e.*, intermediate nodes) from the verified leaf nodes. A recovery of SC-based PAT is impossible because minor counter values at overflow are discarded, which causes uncertainty on the intermediate counters. In contrast to existing mechanisms, Crystalor creates a new tree, where resilience against replay attack is proven, as described in Section 3.3.

3.2 Leaf tag verification using PXOR-Hash

AE can realize the verification of leaf nodes (*i.e.*, payload data) with a nonce consisting of its address and counter. Here, as we use an implicit (*i.e.*, physical) address for the nonce, we can detect a forgery, including splicing, but not a replay attack on a leaf node. To detect a replay, we should verify the integrity of counters using a leaf tag stored on-chip securely. As the leaf tag is verified only after a crash, only its update requires real-time processing (while its verification does not).

[3]Nevertheless, PXOR-Hash is rate-1, which yields a recovery optimal in terms of the number of AES encryption calls.

The requirements of leaf tags for security and practical performance are as follows.

Requirement 1 (Security). *Let F denote a function that computes an n-bit leaf tag from input D, which consists of leaf counter blocks. For any adversary with practical resources, the probability of finding a collision on F (i.e. a distinct pair D and D' such that $F(D) = F(D')$) is negligible in n.*

Requirement 2 (Incremental update for nominal operation). *Assume that old tag and old input blocks are available. If one input block is changed, then the new tag can be computed with $O(1)$ calls of symmetric cryptographic primitive. Note that this assumption is usually true for our context because old data remains on-chip before the update.*

Requirement 3 (Fast recovery). *The tag can be computed and verified with $m + O(1)$ calls of symmetric cryptographic primitive, where m is the input length.*

Requirement 1 is crucial as it directly represents a forgery of a leaf tag given an input (*i.e.*, leaf node counters). The function F is either keyed or unkeyed and in the former case, we assume that the adversary does not know the (random) key. More concretely, if F is keyed, Requirement 1 is equivalent to requiring F to be an almost universal (AU) hash function (See Definition 1) [13]. AU hash functions have been extensively studied, and they can be efficiently constructed by utilizing a secret key dependency, compared to one-way hash functions such as SHA-2 or SHA-3.

For a keyed function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ where $\mathcal{K}$ is the key space, we write F_K to denote $F(K, *)$.

Definition 1 (AU hash function). *Let $F : \mathcal{K} \times \mathcal{D} \to \{0,1\}^n$ be a function for a key $K \in \mathcal{K}$ and plaintext $D \in \mathcal{D}$. The function F is ε-AU hash function if $\Pr[K \leftarrow \mathcal{K} : F_K(D) = F_K(D')] \leq \varepsilon$ holds for any D and $D' \in \mathcal{D}$ such that $D \neq D'$.*

We remark that a full-fledged (nonce-based) MAC will also work; however, an AU hash function is sufficient for our purpose. This is because, in our architecture, the leaf tag is stored on-chip trusted/secure area, where *the adversary in Requirement 1 cannot see nor manipulate it*. This feature is crucial because if the output of the AU hash is visible to the adversary, a collision is usually easy to find. A nonce is unnecessary because *each leaf node counter never repeats under nominal operation*. If the leaf tag was stored off-chip or the plaintext of F could take the same value, we need to employ a conventional MAC, or add a nonce as a new input of F and employ nonce-based MAC. However, this will increase the computational cost or latency compared to an AU hash function. Based on these observations, we develop an AU hash function PXOR-Hash, which fulfills these three requirements. An AU hash function could be built using algebraic operations (e.g., GHASH in GCM); but, such constructions have difficulties in incremental updates for large inputs. Instead, we adopt a computational variant of AU hash function which is a simplified version of PMAC (*i.e.*, sum of input-masked AES). This enables incremental updates for large inputs and provable security guarantee based on the symmetric primitive

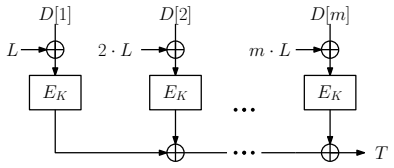


Figure 5: Block diagram of PXOR-Hash.

Algorithm TagGen(D,K,L)
1 $T \leftarrow 0^n$
2 **for** $i = 1$ to m
3 $\quad T \leftarrow E_K(i \cdot L \oplus D[i]) \oplus T$
4 **return** T

Algorithm Verify(D,K,L,T)
1 $T' \leftarrow$ TagGen(D,K,L)
2 **if** $T = T'$
3 $\quad$ **return** $\top$
4 **else**
5 $\quad$ **return** $\bot$

Algorithm Update$(D[i],D'[i],i,K,L,T)$
1 $T \leftarrow E_K(i \cdot L \oplus D[i]) \oplus T$
2 $T \leftarrow E_K(i \cdot L \oplus D'[i]) \oplus T$
3 **return** T

Figure 6: PXOR-Hash algorithms, where $D = (D[1], D[2], \ldots, D[m])$ and $D'[i]$ is new data to be updated.

we use (namely, AES in our case) [60].

Construction of PXOR-Hash. Let $E_K(\cdot)$ denote a block cipher encryption using a secret key K (typically, E_K is AES encryption). Let $D[1], D[2], \ldots, D[i], \ldots, D[m]$ be input data blocks to be verified, where m is the number of data blocks. In the context of Crystalor, each $D[i]$ consists of counters for leaf node nonce (see below). Figure 5 and Figure 6 show the block diagram and algorithmic description of PXOR-Hash, respectively. The tag of PXOR-Hash is computed as

$$T = E_K(L \oplus D[1]) \oplus E_K(2 \cdot L \oplus D[2]) \oplus \cdots \\ \oplus E_K(i \cdot L \oplus D[i]) \oplus \cdots \oplus E_K(m \cdot L \oplus D[m]),$$

where $L = E_K(0)$, operator $\oplus$ denotes a bit-wise XOR, and the multiplication is over $\mathbb{F}_{2^n}$ (n is the block length of E_K). Note that L can be pre-computed and stored in on-chip memory in advance to remove its latency.

Incremental update. Given an old tag T, let us consider updating the i-th block $D[i]$ to $D'[i]$. The new tag is given by

$$T' = E_K(L \oplus D[1]) \oplus E_K(2 \cdot L \oplus D[2]) \oplus \cdots \\ \oplus E_K(i \cdot L \oplus D'[i]) \oplus \cdots \oplus E_K(m \cdot L \oplus D[m]). \quad (1)$$

Using the old tag T and old data $D[i]$, the new tag T' is equivalently computed by

$$T' = T \oplus E_K(i \cdot L \oplus D'[i]) \oplus E_K(i \cdot L \oplus D[i]), \quad (2)$$

which requires only two E_K calls, whereas the naïve computation in Equation (1) requires m calls.

Concrete realization. For the leaf node verification, we compute and store a leaf tag T using PXOR-Hash on-chip, where the inputs are $\mathsf{ctr}[1] \,\|\, \mathsf{ctr}[2] \,\|\, \cdots \,\|\, \mathsf{ctr}[m]$. We use AES-128 for a block cipher, and 64-bit counters (without SC). The input data is given by $D[2i] = \mathsf{ctr}[2i] \,\|\, \mathsf{ctr}[2i+1]$. If the $2i$- or $(2i+1)$-th node is updated, then $D[2i]$ is also updated using the computation in Equation (2). Also, if we use SC, the i-th input data block is given by all node counters related to the i-th major counter; that is, $D[i] = \mathsf{ctr}_{\mathsf{Ma}}[i] \,\|\, \mathsf{ctr}_{\mathsf{mi}}[i][0] \,\|\, \mathsf{ctr}_{\mathsf{mi}}[i][1] \,\|\, \cdots \,\|\, \mathsf{ctr}_{\mathsf{mi}}[i][m']$, where m' is the number of minor counters. If a leaf node related to $\mathsf{ctr}_{\mathsf{Ma}}[i]$ and $\mathsf{ctr}_{\mathsf{mi}}[i][j]$ $(1 \le j \le m)$ is updated, then $D[i]$ is updated as well. Note that address is not required to input to PXOR-Hash because PXOR-Hash can detect a change of block order (*i.e.*, splicing). If a crash occurs, Crystalor first verifies the leaf node using the AE and then detects a replay attack by comparing the on-chip leaf tag and the tag computed by Equation (1).

Security of PXOR-Hash. As mentioned, PXOR-Hash is a computational AU hash function, or more precisely, an almost XOR-universal (AXU) hash function. An AXU hash function is an AU hash function. Note here that, for a leaf node with $\mathsf{ctr}_{\mathsf{Ma}}[i]$ and $\mathsf{ctr}_{\mathsf{mi}}[i][j]$, i and j are implicit inputs to PXOR-Hash (that is, the input order of major and minor counters, which represents the node address). Since PXOR-Hash can detect a swap of bits/blocks, Crystalor is secure against splicing. Thus, PXOR-Hash can detect any manipulation on leaf node counters, if the collision probability is negligible.

If the securely stored hash value (T) and output of PXOR-Hash taking a modified (forged) input collide, integrity will be lost. Concretely, this probability for each forgery attempt is at most $4m/2^n$ when $m \le 2^{n-2}$, where $n = 128$ and m is the number of input blocks, assuming the underlying AES is computationally secure (*i.e.*, a pseudorandom permutation). Hence, the collision probability is negligible in practice if $m \ll 2^{n-2} = 2^{126}$. For example, even for a very large NVM of 1 P bits, the collision probability is less than 2^{-83}, which is practically negligible. The collision probability of PXOR-Hash could be obtained by analyzing (message hashing part of) PMAC [60]. Originally the collision probability was at most $m^2/2^n$ [60], however, Minematsu and Matsushima [50, Lemma 2] improved it to $4m/2^n$, assuming $m \le 2^{n-2}$. These proofs considered doubling-based masks, meaning that the i-th input mask is $2^i \cdot L$, where "2" denotes the generator of the field GF(2^n). This differs from $i \cdot L$ in PXOR-Hash as we adopted for hardware suitability. However, the proof of [50, Lemma 2] also holds for our case, with almost no change. Hence, we omit the proof here.

In case that we want a stronger security bound, which may happen in case m is even larger, or the block size is smaller, we can use stronger methods. For example, (the message hashing part of) PMAC with multiple masks [22] or TBC-based PHASH [60]. The former could be instantiated low-latency block ciphers such as Prince [10], and the latter could be instantiated with a low-latency TBC such as QARMA [4]. Both methods further reduce the contribution of input length in the collision bound.

3.3 NVM recovery by constructing new tree

To date, PAT recovery after a crash has been realized by reconstructing intermediate nodes (*i.e.*, nonce counters) using redundancy or relation between parent and child nodes. For example, Anubis used a shadow table to preserve the node addresses under updates [82]. The SCUE reconstructs intermediate nodes from leaf nodes in a bottom-up manner, owing to the consistency between the sum of child node counters and a parent node counter. These existing methods cannot work with SC because minor counter values are discarded and reset when an overflow occurs.

Here, intermediate nodes are not payload data but only for verifying the leaf nodes regarding a replay attack. In other words, the intermediate nodes are unnecessary if we can verify the leaf nodes in another way (*e.g.*, the leaf tag of PXOR-Hash). Therefore, Crystalor relinquishes the old tree except for the leaf node but constructs a new tree instead. However, if an old counter is used in the new tree, it is exploited by a replay attack, resulting in a feasible forgery. Accordingly, we should construct a new tree with counter values greater than the old ones for resilience against replay.

Our idea is to use an upper bound of the number of updates as the new counter. Consider a case in which k leaf nodes share a major counter and the minor counter bit length is l. A parent node has a major counter $\mathsf{ctr}_{\mathsf{Ma}}[i]$ and a minor counter $\mathsf{ctr}_{\mathsf{mi}}[i][j]$ for each $1 \le j \le k$. For a PAT with arity of β, it has β child nodes with major counters $\mathsf{ctr}_{\mathsf{Ma}}[i']$ and minor counters $\mathsf{ctr}_{\mathsf{mi}}[i'][j']$ ($1 \le i' \le \beta/k$ and $1 \le j' \le k$). After a crash, Crystalor computes the possible maximum number of the parent nodes, denoted by $\mathsf{ctr}_{\mathsf{pa}}[i][j]$, as

$$\mathsf{ctr}_{\mathsf{pa}}[i][j] = \sum_{i'=1}^{\beta/k} \left(\mathsf{ctr}_{\mathsf{Ma}}[i'] \left(k(2^l-1)+1 \right) + \sum_{j'=1}^{k} \mathsf{ctr}_{\mathsf{mi}}[i'][j'] \right),$$

Crystalor then computes the major and j-th minor counter values of i-th intermediate node as

$$\mathsf{ctr}_{\mathsf{Ma}}[i] = \sum_{j=1}^{k} \left\lfloor \frac{\mathsf{ctr}_{\mathsf{pa}}[i][j]}{2^l} \right\rfloor, \tag{3}$$

$$\mathsf{ctr}_{\mathsf{mi}}[i][j] = \mathsf{ctr}_{\mathsf{pa}}[i][j] \bmod 2^l, \tag{4}$$

respectively, where $\lfloor \cdot \rfloor$ is the floor function. All counters of intermediate and root nodes are computed bottom-up by repeating this computation from the leaf nodes. This represents that the upper bits of nonce are shared as a major counter and lower l bits are unique to each minor counter.

We prove Proposition 1 to validate the security of Crystalor's recovery against replay attack.

Proposition 1. *Let* $\mathsf{ctr}_{\mathsf{Ma}}[i]$ *and* $\mathsf{ctr}_{\mathsf{mi}}[i][j]$ *be the i-th parent node major and minor counter values computed by Equations* (3) *and* (4)*. Any new tree is resistant to replay attacks.*

Proof. First, we consider a single crash. Let c be the number of updates from the previous reset of minor counters until the next reset (*i.e.*, major counter increment). It always holds $2^l - 1 \le c \le k(2^l-1)$ because c is minimum if only one node is updated (and others are not updated at all), while c is maximum if each of k minor counters has the maximum value (*i.e.*, 2^l-1). Let $u_{i'}$ be the total number of updates of nodes sharing the i'-th major counter, which is bounded above as

$$u_{i'} \le \mathsf{ctr}_{\mathsf{Ma}}[i'] \left(k(2^l-1)+1 \right) + \sum_{j'=1}^{k} \mathsf{ctr}_{\mathsf{mi}}[i'][j'],$$

because $\mathsf{ctr}_{\mathsf{Ma}}[i']$ denotes the number of minor counter resets. Let $u_{i,j}$ be the number of updates of a parent node with $\mathsf{ctr}_{\mathsf{mi}}[i][j]$, which is bounded as

$$u_{i,j} \le \sum_{i'=1}^{b} u_{i'} = \mathsf{ctr}_{\mathsf{pa}}[i][j].$$

This indicates that $\mathsf{ctr}_{\mathsf{pa}}[i][j]$ is greater than or equal to the number of updates of the node (*i.e.*, the true value of parent counter ever before). The equality holds if c has taken the maximum whenever and wherever the minor counter resets or if any minor counter reset has not occurred (*i.e.*, $\mathsf{ctr}_{\mathsf{Ma}}[i'] = 0$ for all i'). Thus, for all $1 \le j \le k$, the parent node was updated at most $\mathsf{ctr}_{\mathsf{pa}}[i][j]$ times. Because it also holds for the parent node that $2^l - 1 \le c \le k(2^l-1)$, the number of updates of the parent major counter must be less than $\sum_{j=1}^{k} \lceil \mathsf{ctr}_{\mathsf{pa}}[i][j]/2^l \rceil$, which indicates the new major counter value never appears before the crash. Thus, its replay is impossible.

We then consider multiple crashes, in which counter values are given by Equations (3) and (4) in the past. If a leaf node counter is incremented, then a corresponding $\mathsf{ctr}_{\mathsf{pa}}[i]$ always takes a greater value than the previous state, because it is strictly monotonically increasing in terms of both $\mathsf{ctr}_{\mathsf{Ma}}[i']$ and $\mathsf{ctr}_{\mathsf{mi}}[i'][j']$. This implies that either or both $\mathsf{ctr}_{\mathsf{Ma}}[i]$ and $\mathsf{ctr}_{\mathsf{mi}}[i][j]$ in Equations (3) and (4) are greater than any previous state. In addition, if a child node is updated, its parent node counter accordingly increases; however, its increase amount is not as great as the number of updates as abovementioned. Thus, the new counter values determined by Equations (3) and (4) are always new, which guarantees the resistance to replay attacks. □

The integrity of leaf nodes is verified by AE, excluding replay attacks. The leaf tag verification detects the replay attacks on leaf nodes. Moreover, Proposition 1 states that the counter values of the new tree are always greater than values before the crash, indicating the new tree's resistance to replay attacks. Thus, Crystalor provides both crash recoverability and integrity against any manipulation attacks. Note that, in a recovery operation, the new tree construction and leaf node/tag verification should be carefully executed in such a way as to avoid replay attacks (see Section 3.5).

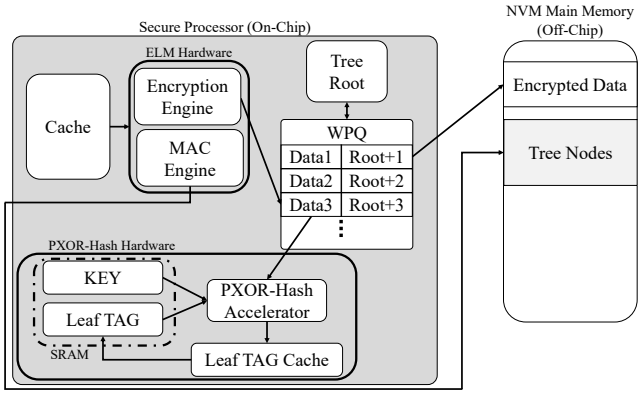


Figure 7: Crystalor hardware architecture.

3.4 Hardware architecture

Figure 7 displays the hardware architecture of Crystalor for persistent memory encryption [32], in which we employ ELM. Crystalor utilizes dedicated hardware components to compute and update the leaf tag apart from memory protection hardware for ELM computation, as Crystalor operates distinctly and independently of PAT. The dedicated hardware consists of an SRAM to store and update the leaf tag (Leaf TAG register) in addition to the secret key of PXOR-Hash key (KEY register). PXOR-Hash hardware consists of pipelined AES encryption hardware, which can process multiple update transactions in parallel in the most efficient manner. The leaf tag is stored in both the SRAM and cache (Leaf TAG register and cache) to improve the speeds of tag computation. This dedicated hardware operates at every timing of leaf node update (*i.e.*, storing encrypted payload data to NVM) to simultaneously and consistently update/store the leaf tag to Leaf TAG register and cache. In Figure 7, the leaf tag is always updated on-chip but is not disclosed to the off-chip NVM. This is mandatory for security to prevent any manipulation attack on these data. In other words, as the leaf tag is securely processed and stored, it does not require protection as strong as MAC, leading to an efficient implementation of PXOR-Hash.

Crystalor requires an on-chip SRAM for storing the 128-bit leaf tag and PXOR-Hash keys (K and L). So, the SRAM overhead is 384 ($= 128 \times 3$) bits in total. Crystalor also utilizes a 128-bit on-chip cache for Leaf TAG cache and the AES encryption engine in PXOR-Hash hardware which is implemented with less than 15 K GE [68].

Furthermore, the proposed architecture has an on-chip non-volatile register to store temporal data during ELM (*i.e.*, AE and MAC) computation. When the ELM computation for the temporal data is completed, the architecture raises a flag bit to start writing the data to the WPQ. If a system crash occurs with the frag raised, the on-chip non-volatile register data is written to the WPQ after the system reboots. Otherwise (*i.e.*, without the flag raised), data in the register is discarded.

The remaining parts other than Crystalor operate similarly to the conventional ones. A memory controller controls the NVM data to handle encrypted payload data and security metadata. The ELM hardware includes a 952-bit cache for storing the ELM secret key and its precomputable intermediate values. We employ an on-chip WPQ to persist data during store operation using ADR [64]. Namely, as leaf nodes must be consistent with the on-chip leaf tag, we explicitly use a WPQ as an ADR domain for the leaf node to guarantee its persistence. Note that the intermediate nodes are discarded at a crash, hence they do not require persistency; thus, MAC outputs (*i.e.*, intermediate nodes) are directly written to NVM without using WPQ (which indicates the intermediate nodes are computed and updated with background processing like SCUE). Furthermore, we utilize atomic persistency mechanism(s) and hardware redo logging as in existing methods.

Remark 3.1 (Combination with other mechanisms). Figure 7 presents the simplest construction without any optimization mechanism. Advanced counter mechanisms such as SC, Vault, and Morphable counters are applicable. Adoption of mechanism(s) for atomic data persistency would also be essential for consistent transactions with high performance [15,28,33,35,46,54,63,71,78,80]. As PXOR-Hash distinctly operates from PAT, such optimization mechanisms for PAT can be readily incorporated together.

Performance overhead. The latency overhead of Crystalor during nominal operation does *not* depend on the cache size nor covered NVM region, but solely depends on the computational cost of leaf tag update. As mentioned in Section 3.2, the leaf tag update is completed within only two AES encryption calls for new and old $D[i]$'s. The two AES encryptions are performed in parallel using pipelined AES hardware, which requires far smaller latency of ELM update. Therefore, the latency overhead of Crystalor is negligible and has no impact on the system performance, as Crystalor and PAT distinctly operate. Owing to the simplicity of Crystalor, it is easy to implement and incurs little overhead in the memory controller.

3.5 Operations

We describe the store and recovery operations of ELM–Crystalor. Read operation requires no Crystalor operation.

Store operation. For crash recoverability and security, we must simultaneously and consistently update the leaf node counter in the NVM and the on-chip leaf tag, which is realized by the following steps:

1. Store operation to the NVM is issued. It would correspond to a cache data replacement (*i.e.*, cache miss) or an explicit instruction to guarantee the data persistency at a timing, such as `clflush` and `clwb` in x86 architecture.
2. (**ELM computation**) Payload data is encrypted, corresponding intermediate nodes and root node tags are computed in the background, and the corresponding counters

are incremented. The results are stored in the on-chip non-volatile register.

3. (**PXOR-Hash computation**) The new leaf tag is computed from the old tag, old data, and new data.
4. (**Write to WPQ**) The non-volatile register data is moved to WPQ. When leaf node is in the WPQ, a busy flag for this transaction (on-chip non-volatile register) is raised.
5. (**Tag update**) Leaf TAG cache and registers are updated.
6. (**Store data**) WPQ data is stored in NVM, and the root node counter is incremented. The busy flag is put down when completed.

Note that Steps (2, 3) and (4, 5) should be synchronously executed in parallel for consistency between the leaf tag and leaf counters. In addition, data should be updated in an atomically persistent manner, with the help of some mechanisms for this purpose (*e.g.*, [15, 28, 33, 35, 46, 54, 63, 71, 78, 80]).

Recovery. After a crash and system reboot, Crystalor securely recovers the NVM as follows:

1. (**WPQ data reflection**) If the busy flag has been raised, the WPQ data is written to the NVM; otherwise, it is discarded. Then, the WPQ data is reflected in the NVM based on the redo logs. We here suppose that the data are atomically persistent, as mentioned before.
2. (**New tree construction**) According to Section 3.3, counter values of intermediate and root nodes are derived in a bottom-up manner. The MAC tag of each node is also computed from the counters and addresses.
3. (**Leaf tag verification**) The verification leaf tag of PXOR-Hash is computed from all leaf node counters, and then the computed tag is compared to the on-chip Leaf TAG register value. If they are equivalent, the nominal operation restarts; otherwise, Crystalor detects a manipulation and gives an error signal.

We should perform Steps 2 and 3 in this order to prevent replay attacks. Namely, we create a new tree, although leaf nodes may be manipulated. Hereafter, PAT can detect any manipulation, while it cannot detect leaf node manipulation during a crash. Then, we verify the leaf tag to detect a replay of the leaf node. Thus, the attacker cannot manipulate after the new tree construction. In contrast, all manipulations before the new tree construction are detected by the leaf tag verification and the subsequent leaf node AE verification (lazy verification/recovery). Note that if we execute Steps 2 and 3 in the reverse order, a replay attack is possible after the leaf tag verification before starting new tree construction.

On lazy verification/recovery. Crystalor can detect any counter replay solely by the leaf tag verification, while SCUE cannot[4]. Hence, at a reboot, Crystalor does *not* require verifying the AE leaf node for protection because it will be verified before it is actually used. In other words, the leaf node AE verification can be omitted at the time of recovery, and the verification is completed lazily and concurrently during nominal operation after the recovery. Thus, Crystalor does not have to read the leaf node data nor compute AE for the leaf node at reboot time. This yields a significant reduction of recovery cost (as evaluated in Section 4.3) compared to the non-lazy recovery of SCUE because the leaf node occupies a major part of secure NVM. Such a lazy strategy was taken in some previous sutdies [82], and is considered practically secure. An attacker in a practical use scenario of memory encryption, who can manipulate off-chip data and can trigger crashes, cannot bypass the leaf tag verification and PAT verification simultaneously, while their combination detects any replay.

[4]At a crash, an attacker can insert a replay data without SCUE detected by incrementing another leaf node counter, such that the sum of leaf node counters is preserved. If the replayed node is loaded before detecting incremented counters, the replay attack is not detected and succeeded. Thus, whole leaf node AE verification is essential to detect such a replay with a counter increment. This implies the insecurity of lazy recovery for SCUE. Therefore, SCUE cannot use such a lazy strategy and essentially requires verifying the leaf node AE at the timing of recovery for security against replay attacks.

4 Performance evaluation

4.1 Algorithm-level evaluation

In the following, we consider the typical SC parameter [31, 55]: the lengths of major and minor counters (*i.e.*, l_{Ma} and l_{mi}) are given by 56 and 8 bits, respectively, and the number of nodes sharing a major counter (*i.e.*, k) is 8.

SC was originally proposed as an optimization method to reduce the metadata size (*i.e.*, NVM overhead). The size reduction of metadata may also contribute to a latency reduction because the length of data to be verified by MAC is also reduced. We derive the relation between the covered region size and ELM parameters to calculate the (optimal) latency of ELM for a given covered region size. Then, we can choose an optimal parameter that covers the region with the minimum latency. Note here that the advantages of SC directly represent the supremacy of Crystalor over SCUE.

Latency and covered region size. Let b denote the number of input blocks to MAC and let ℓ denote the bit length of a leaf node. Here, b is derived from the tree arity β as $b = \beta/2$ and $b = \beta/8$ without and with SC, respectively. Without SC, an intermediate node has $2b$ child nodes because a counter is given by 64 bits. This indicates that the tree has $2^d b^d$ leaf nodes. If SC is applied, an intermediate node can have $8b$ nodes because an input block $D[i]$ consists of a 64-bit major counter and 8-bit minor counters of eight child nodes. This indicates that the tree with SC has $2^{3d} b^d$ leaf nodes. For given b and ℓ, ELM can cover a region of $2^d b^d \ell$ and $2^{3d} b^d \ell$ bits without and with SC, respectively. Meanwhile, the update latencies of Flat-OCB and PXOR-MAC hardware in [32] are $14 + \ell/128$ and $12 + b$ clock cycles, respectively. Note that the metadata size does not include bits to indicate the address

Table 1: Latency and covered region of ELM with and without SC, where b is number of input blocks (corresponding to tree arity), ℓ is bit length of AE, and d is tree depth

				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

† "Update" actually means "Verify then Update," because PAT requires tag verification always before update for security [14, 32].

because it is implicit. Table 1 reports the latency and covered region size for different values of b and ℓ without and with SC, where the latency means $\min(14+\lceil \ell/128 \rceil, 12+b)$ as the bottleneck. From Table 1, we confirm that SC significantly reduces the latency to cover a given region. For example, for $d=5$ and 7, to cover a 4 TB region, ELM without SC requires at least 78 and 30 clock cycles for the update, whereas ELM with SC requires only 30 and 22 clock cycles. Thus, SC reduces the latency overhead by 38 and 8 cycles for $d=5$ and 7 (*i.e.*, 62% and 29%).

Metadata size. We evaluate how much SC contributes to reducing NVM overhead to store metadata. Without SC, the metadata size is given by $112\sum_{i=0}^{d}\beta^i - 56$, whereas, with SC, it is $72\sum_{i=0}^{d}\beta^i - 56\sum_{i=0}^{d-1}\beta^i$, according to [31]. For example, to cover a 4 TB region with a tree of $b=4$, ELM without and with SC has overhead of 554 GB and 312 GB to store the metadata, respectively, which indicates a 44% reduction of the overhead by SC. Thus, SC significantly improves the performance of memory encryption.

Additional latency due to minor counter overflow. When a major counter is incremented (*i.e.*, minor counter overflows), SC requires the re-computation of tags related to the major counter. If the j-th node of i-th major counter overflows, then the major counter $\mathsf{ctr}_{\mathsf{Ma}}[i]$ is incremented, and the minor counters $\mathsf{ctr}_{\mathsf{mi}}[i][j]$ for all j $(1 \le j \le k)$ are reset to zero. We should recompute the tag of nodes for all j, as its nonce counter $\mathsf{ctr}_{\mathsf{Ma}}[i] \,\|\, \mathsf{ctr}_{\mathsf{mi}}[i][j]$ is updated. This means that $k-1$ MAC/AE updates accompany a minor counter overflow. The system-level simulation for the performance evaluation should regard the latency due to minor counter overflow. Nevertheless, the latency overhead by minor counter overflow is not critical as its frequency is low. On average, it incurs less than one clock cycle latency per store operation.

Remark 4.1 (Tree depth and hardware resource). Tree depth d is a parameter that exploits tradeoffs between a *hardware resource* (*i.e.*, the number of MAC engines) and covered region/latency, while b and ℓ optimal in terms of latency are determined systematically for a given covered region. In other words, for an optimal fixed b and ℓ, we can enlarge the covered region size by increasing d, using $d-1$ parallel MAC engines. Conversely, we can reduce the latency for a fixed covered size by increasing d. In other words, the significance of covered region size and improvement by SC depends on d.

4.2 System-level simulation

We perform system-level simulations of secure NVM using the gem5 simulator [9] for the validation. In this simulation, we evaluate the crash mechanisms with ELM, as ELM is the state-of-the-art and achieves the highest performance among PATs[5]. We assume to utilize AE and MAC hardware presented in [21, 32] for the ELM hardware in this evaluation. We evaluate the proposed and existing methods as follows:

- Insecure: NVM without any security mechanism.
- ELM w/o SC: ELM without SC (Not recoverable).
- ELM with SC: ELM with SC (Not recoverable).
- ELM–SCUE: ELM with SCUE (SC is inapplicable).
- ELM–ASIT: ELM with Anubis (SC is inapplicable).
- ELM–Crystalor (this work): ELM with Crystalor, to which SC is applied.

Insecure and ELMs (Not recoverable) are the baselines to evaluate the overhead of PAT and crash recoverability, respectively. We determined the latency according to the memory capacity (*i.e.*, 4 TB) and Table 1. For the ease, feasibility, and reproducibility of the experiment, we employed several

[5] Some previous studies (*e.g.*, [30]) utilized a classical HMAC, which is assumed to require 40, 80, or 160 clock cycles for Verify and Update. However, its concrete realization/implementation is not mentioned, and the number of input blocks to AE/MAC is not considered, although they determine the latency; thus, its practical validity and feasibility are unclear. We employ the ELM-style evaluation to determine the clock cycles for a fair, modern, and practical performance comparison, different from previous studies. Our results are based on the in-depth evaluation of latency in the ELM paper, which considers a concrete cryptographic hardware implementation and the number of input blocks to AE/MAC, while the previous studies did not. Note that HMAC is not optimal in terms of latency and incrementality, and therefore PXOR-MAC in ELM was proposed for an optimized latency [32].

Table 2: Simulation conditions

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
Memorey controller and NVM	
WPQ size	8 entries
NVM latency	Read 50 ns and Write 150 ns
NVM 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 $\ell = 1{,}024$

simplifications for the simulation and previous studies. We omitted the simulation of packet metadata written to NVM. We virtually inserted the latency to write and store operations due to ELM according to Table 1. For ELM with SC, to evaluate the latency about minor counter overflow, we employed an apportionment, in which we assume that the writings to NVM are uniformly distributed, calculate the expected latency due to the minor counter overflow, and add the rounded-up value to the latency in Table 1. These simplifications are applied to all the above methods, allowing for a fair and sound comparison in addition to reproducibility.

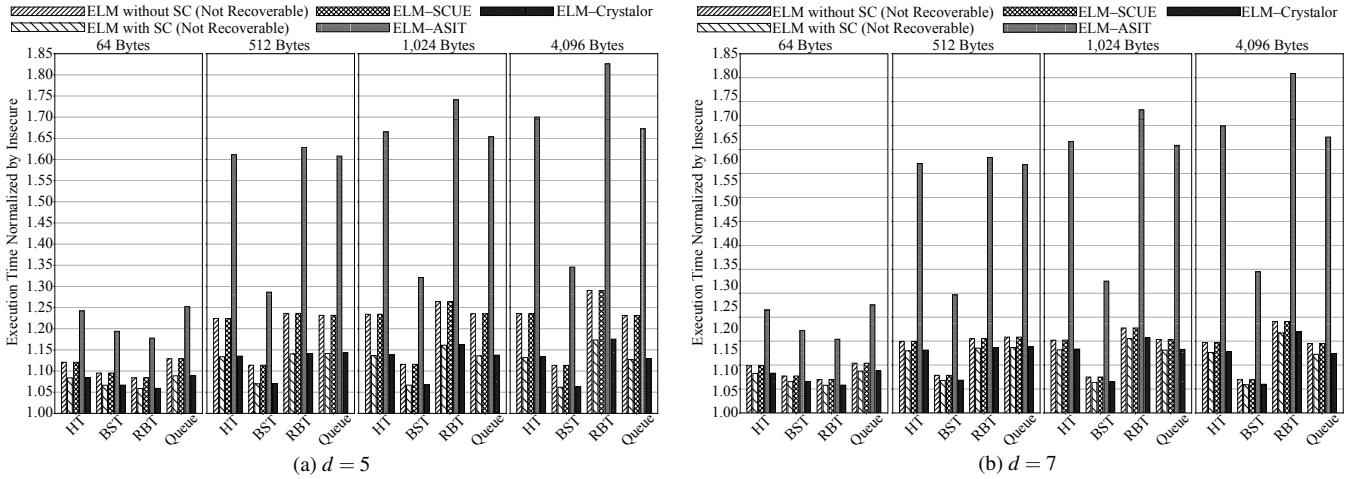
We employ a benchmarking workload set, which has been commonly used in many previous studies on secure and recoverable NVM like a de facto standard (*e.g.*, [29, 39, 75, 84]). The workloads include random insertions of data to a hash table (HT), binary search tree (BST), red-black tree (RBT), and queue (Queue), each of which has a distinct memory access pattern. To analyze the difference, we simulate the workloads with data sizes of 64, 512, 1,024, and 4,096 Bytes.

Results. Figure 8 reports the normalized workload execution times of the gem5 simulation, in which Insecure is the baseline. We do not evaluate $d = 3$, as ELM with $d = 3$ without SC cannot cover a 4 TB region with a practical latency (this shows SC's significance). ELM–Crystalor has almost the same performance as ELM with SC what ELM–SCUE is to ELM without SC. Because Crystalor and SCUE incur no latency overhead under nominal operation, the performance of ELM–Crystalor and ELM–SCUE depends solely on the tree parameter. In contrast, ASIT incurs a non-trivial latency overhead to verify and update the shadow table. Comparing ELM with and without SC (*i.e.*, ELM–Crystalor and ELM–SCUE), the performance gain by SC is more significant when the data size is larger. This is because the reduction of latency in reading and writing (*i.e.*, verifying and updating) NVM data is more dominant and visible as the numbers of data read and write increase for the larger data size. In addition, the improvement in execution time by SC is more significant when $d = 5$ than $d = 7$, because the reduction ratio of latency by SC is larger when $d = 5$ for covering a 4 TB region. In consequence, we confirm that ELM–Crystalor can reduce the workload execution time by at most 11.5% than the state-of-the-art mechanism (*i.e.*, ELM–SCUE).

Improved scalability for larger NVM. Regarding the tree depth d, the performance gain by the proposed method is greater for a shorter tree (*i.e.*, $d = 5$ in this experiment). Recall that d is a parameter that exploits tradeoffs between a hardware resource (*i.e.*, the number of MAC engines) and a covered region. As the cover region size in the experiment is fixed as 4 TB, the size is relatively larger for $d = 5$, and the latency overhead by PAT-based protection is larger for $d = 5$. The SC compresses the tree/metadata size more effectively when protecting a larger NVM. Hence, the performance gain by SC (and the proposed method) is greater for $d = 5$. More quantitatively, for a given arity β, the use of SC can reduce the number of input blocks to PXOR-MAC to $1/4$ in the used parameter. This indicates that the use of SC reduces the latency of PXOR-MAC asymptotically by $1/4$ for a larger β. Thus, the use of SC (and ELM-Crystalor) reduces the latency of NVM read/write to up to $1/4$ when protecting a larger NVM. The experimental results on $d = 5$ and 7 indicate the proposed method's improved scalability.

4.3 Recovery cost estimation

Lazy recovery cost of Crystalor. We consider here an ELM–Crystalor with SC, whose major and minor counters are 56 and 8 bits, respectively. The recovery time of Crystalor is evaluated by the number of AES calls for PXOR-hash of leaf tag verification and PXOR-MAC of new tree construction. To protect an M-bit NVM, the number of AES calls in PXOR-Hash is equivalent to $M/8\ell$, while the leaf node verification by Flat-OCB is not required at the time of recovery as mentioned in Section 3.5. In addition, for arity of β, a PXOR-MAC computation requires $1+\beta/8$ AES calls, while a new tree construction is realized with $\sum_{i=1}^{d}\beta^{i-1}$ PXOR-MAC computations. This indicates that the new tree construction requires $(1+\beta/8)\sum_{i=1}^{d}\beta^{i-1}$ AES calls in total. Moreover, a new tree construction requires computations of new counter value $\sum_{i=1}^{d}\beta^{i-1}$ times. Recall that $M = \beta^d\ell$. Thus, Crystalor recovery is realized with $\beta^d/8 + (1+\beta/8)\sum_{i=1}^{d}\beta^{i-1}$ AES calls and $\sum_{i=1}^{d}\beta^{i-1}$ new counter computations, which corresponds to the number of intermediate nodes (including root node). In addition, Crystalor requires to read $8\beta^d$ bits from NVM, where $8\beta^d$ bits are leaf node metadata, while Crystalor writes $128(1/2+\beta/8)\sum_{i=2}^{d}\beta^{i-1}$ bits of metadata to NVM during a recovery. Note that, according to the lazy reduction,



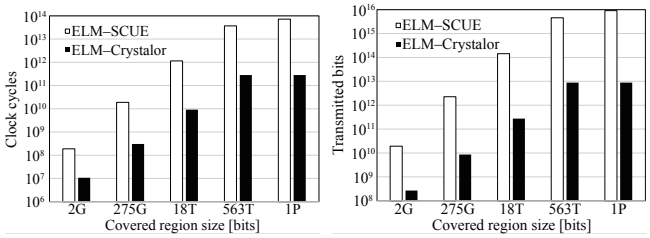
(a) $d = 5$ (b) $d = 7$

Figure 8: Simulated execution times normalized by Insecure, where proposed method is right-most bin.

the costs are independent of the leaf node bit length (*i.e.*, ℓ).

Recovery cost of SCUE. For comparison, we here consider ELM–SCUE recovery with 64-bit counter. The SCUE recovery cost is evaluated by the number of counter-summing and PXOR-MAC computations. For the equivalence check between the root counter and the sum of leaf node counters, $\sum_{i=1}^{d} \beta^{i-1}$ counter-summings are required. Then, the tree recovery performs one counter-summing and one PXOR-MAC computation per node, while a PXOR-MAC requires $1+\beta/2$ AES calls. In addition, there are $\sum_{i=1}^{d} \beta^{i-1}$ intermediate nodes. The leaf node AE verification requires $\beta^d(\lceil \ell/128 \rceil + 1)$ AES calls. Thus, the computational cost is $2\sum_{i=1}^{d} \beta^{i-1}$ counter-summings and $\beta^d(\lceil \ell/128 \rceil + 1) + (1+\beta/2)\sum_{i=1}^{d} \beta^{i-1}$ AES calls. Meanwhile, SCUE reads $\beta^d \ell + 64\beta^d$ bits from NVM, while it writes $128(1/2+\beta/2)\sum_{i=2}^{d} \beta^{i-1}$ bits to NVM.

Evaluation result. Figure 9a and Figure 9b report the computational cost (*i.e.*, the number of clock cycles) and the traffic cost between CPU and NVM for some covered region sizes, respectively. Here, the throughputs of counter-summing, new counter computation, and AES encryption are supposedly one per clock cycle [14, 32]. In Figure 9b, the traffic cost is evaluated by the number of transmitted bits, while the writing cost is tripled as in Table 2. Crystalor achieved a reduction of 10–1000% recovery costs from SCUE, thanks to the lazy recovery. Although the computational and traffic costs of the leaf node AE verification is a major part of SCUE (*i.e.*, $\beta^d(\lceil \ell/128 \rceil + 1)$ AES calls and $\beta^d \ell$ bits read, respectively), Crystalor does not require them. Thus, we confirm the advantage of Crystalor in recovery cost as well as the performance.



(a) Computational cost (b) Traffic cost

Figure 9: Estimation of recovery costs.

5 Conclusion

This study presented Crystalor, a persistent memory encryption mechanism with a structural optimization such as SC. Crystalor incurs almost no latency overhead under nominal operation and achieves an efficient recovery. Although existing state-of-the-art mechanisms (*e.g.*, SCUE) are incompatible with structural optimizations, Crystalor fully exploits its advantages and offers the same security and recoverability. We confirmed both algorithmically and experimentally that Crystalor has a significant advantage in the memory overhead and execution time/latency over conventional mechanisms with a reduced recovery cost. At the algorithmic level, for protecting a 4 TB memory with ELM, Crystalor requires 29–62% fewer clock cycles per memory read/write operation than SCUE, while Crystalor and SCUE need 312 GB and 554 GB memory overheads for storing the security metadata, respectively (namely, Crystalor achieves a reduction of NVM overhead by 44%). We then performed a system-level simulation using the gem5 simulator. We confirmed that Crystalor achieves a reduction of workload execution time by at most 11.5% from SCUE. Moreover, Crystalor can offer a lazy recovery owing to its cryptographic protection, which achieved a 10–1000 times faster recovery than SCUE.

We employed the SC as the most typical optimization technique for PAT; however, Crystalor can work with any structural optimization as it employs cryptographic protection. Its application and evaluation with other optimization techniques (in Section 1.3) are important future work.

References

[1] AMD secure encrypted virtualization (SEV). https://www.amd.com/en/developer/sev.html. Visited in September 2023.

[2] Intel Optane technology. https://www.intel.com/content/www/us/en/architecture-and-technology/optane-technology/optane-for-data-centers.html. Visited in September 2023.

[3] Mazen Alwadi, Kazi Abu Zubair, David Mohaisen, and Amro Awad. Phoenix: Towards ultra-low overhead, recoverable, and persistently secure NVM. *IEEE Transactions on Dependable and Secure Computing*, 19(2):1049–1063, 2020.

[4] Roberto Avanzi. The QARMA block cipher family. almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Trans. Symmetric Cryptol.*, 2017(1):4–44, 2017.

[5] Roberto Avanzi, Subhadeep Banik, Orr Dunkelman, Hector Montaner, Prakash Ramrakhyani, Francesco Regazzoni, and Andreas Sandberg. Protecting memory contents on ARM cores. Real World Crypto (RWC), 2020.

[6] Amro Awad, Mao Ye, Yan Solihin, Laurent Njilla, and Kazi Abu Zubair. Triad-NVM: Persistency for integrity-protected and encrypted non-volatile memories. In *ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA '19)*, pages 104–115, 2019.

[7] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography and application to virus protection. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, page 45–56, New York, NY, USA, 1995. Association for Computing Machinery.

[8] Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *Journal of Cryptology*, 21:469–491, 2008.

[9] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[10] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[11] Robert Buhren, Hans-Niklas Jacob, Thilo Krachenfels, and Jean-Pierre Seifert. One glitch to rule them all: Fault injection attacks against AMD's secure encrypted virtualization. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2875–2889, New York, NY, USA, 2021. Association for Computing Machinery.

[12] Federico Canale, Tim Güneysu, Gregor Leander, Jan Thoma, Yosuke Todo, and Rei Ueno. SCARF: A low-latency block cipher for secure cache-randomization. In *32nd USENIX Security Symposium*, 2023.

[13] J. Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–0154, 1979.

[14] Victor Costan and Srinivas Devadas. Intel SGX explained. Cryptology ePrint Archive, Paper 2016/086, 2016.

[15] Nai-Jia Dong, Hsiang-Yun Cheng, Chia-Lin Yang, Bo-Rong Lin, and Hsiang-Pang Li. Efficient and atomic-durable persistent memory through in-pm hybrid logging. In *2022 IEEE 11th Non-Volatile Memory Systems and Applications Symposium (NVMSA)*, pages 1–7, 2022.

[16] Morris J. Dworkin. SP 800-38A 2001 edition. Recommendation for block cipher modes of operation: Methods and techniques. Technical report, National Institute of Standards & Technology, Gaithersburg, MD, USA, 2001.

[17] Morris J. Dworkin. SP 800-38B. Recommendation for block cipher modes of operation: The CMAC mode for authentication. Technical report, Gaithersburg, MD, USA, 2005.

[18] Tetsuo Endoh, Hiroaki Honjo, Koichi Nishioka, and Shoji Ikeda. Recent progresses in STT-MRAM and SOT-MRAM for next generation MRAM. In *2020 IEEE Symposium on VLSI Technology*, pages 1–2, 2020.

[19] Alexander Freij, Shougang Yuan, Huiyang Zhou, and Yan Solihin. Persist level parallelism: Streamlining integrity tree updates for secure persistent memory. In *2020 53rd Annual IEEE/ACM International Symposium*

on Microarchitecture (MICRO), pages 14–27, October 2020.

[20] Alexander Freij, Huiyang Zhou, and Yan Solihin. Bonsai Merkle Forests: Efficiently achieving crash consistency in secure persistent memory. In *54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1227–1240, 2021.

[21] Blaise Gassend, G. Edward Suh, Dwaine Clarke, Marten van Dijk, and Srinivas Devadas. Caches and hash trees for efficient memory integrity verification. In *the 9th International Symposium on High-Performance Computing Architecture*, 2003.

[22] Peter Gazi, Krzysztof Pietrzak, and Michal Rybár. The exact security of PMAC. *IACR Trans. Symmetric Cryptol.*, 2016(2):145–161, 2016.

[23] Hannes Gross, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *ACM Workshop on Theory of Implementation Security (TIS 2016)*, page 3, 2016.

[24] Shay Gueron. A memory encryption engine suitable for general purpose processors. Cryptology ePrint Archive, Paper 2016/204, 2016. https://eprint.iacr.org/2016/204.

[25] Shay Gueron. Memory encryption for general-purpose processors. *IEEE Secur. Priv.*, 14(6):54–62, 2016.

[26] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Parl, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: cold-boot attacks on encryption keys. *Communication of the ACM*, 52:91–98, 2009.

[27] W. Eric Hall and Charanjit S. Jutla. Parallelizable authentication trees. In *Selected Areas in Cryptography*, pages 95–109. Springer Berlin Heidelberg, 2006.

[28] Xijing Han, James Tuck, and Armo Awad. Dolos: Improving the performance of persistent applications in ADR-supported secure memory. In *54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1241–1253, 2021.

[29] Jianming Huang and Yu Hua. A write-friendly and fast-recovery scheme for security metadata in non-volatile memories. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 359–370, Seoul, Korea (South), February 2021. IEEE.

[30] Jianming Huang and Yu Hua. Root crash consistency of SGX-style integrity trees in secure non-volatile memory systems. In *Proceedings of the 29th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2023.

[31] Akiko Inoue, Kazuhiko Minematsu, Maya Oda, Rei Ueno, and Naofumi Homma. ELM: A low-latency and scalable memory encryption scheme. Cryptology ePrint Archive, Paper 2020/1374, 2020. Preliminary and long version of a paper with same title.

[32] Akiko Inoue, Kazuhiko Minematsu, Maya Oda, Rei Ueno, and Naofumi Homma. ELM: A low-latency and scalable memory encryption scheme. *IEEE Transactions on Information Forensics and Security*, 17:2628–2643, 2022.

[33] Jungi Jeong, Chang Hyun Park, Jaehyuk Huh, and Seungryoul Maeng. Efficient hardware-assisted logging with asynchronous and direct-update for persistent memory. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 520–532, 2018.

[34] Arpit Joshi, Vijay Nagarajan, Stratis Viglas, and Marcelo Cintra. ATOM: Atomic durability in non-volatile memory through hardware logging. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 361–372, February 2017. ISSN: 2378-203X.

[35] Arpit Joshi, Vijay Nagarajan, Stratis Viglas, and Marcelo Cintra. ATOM: Atomic durability in non-volatile memory through hardware logging. In *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 361–372, 2017.

[36] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (S&P)*, pages 1–19, 2019.

[37] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO 1999*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[38] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology—CRYPTO 1996*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[39] Mengya Lei, Fan Li, Fang Wang, Dan Feng, Xiaomin Zou, and Renzhi Xiao. SecNVM: An efficient and write-friendly metadata crash consistency scheme for secure

NVM. *ACM Transactions on Architecture and Code Optimization*, 19(1):1–26, March 2022.

[40] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. A systematic look at ciphertext side channels on AMD SEV-SNP. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 337–351, 2022.

[41] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. Exploiting unprotected I/O operations in AMD's secure encrypted virtualization. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1257–1272, Santa Clara, CA, August 2019. USENIX Association.

[42] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. CIPHERLEAKS: Breaking constant-time cryptography on AMD SEV via the ciphertext side channel. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 717–732. USENIX Association, August 2021.

[43] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990, Baltimore, MD, August 2018. USENIX Association.

[44] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy*, pages 244–256. IEEE, 2015.

[45] Sihang Liu, Aasheesh Kolli, Jinglei Ren, and Samira Khan. Crash consistency in encrypted non-volatile main memory systems. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 310–323, February 2018. ISSN: 2378-203X.

[46] Sihang Liu, Aasheesh Kolli, Jinglei Ren, and Samira Khan. Crash consistency in encrypted non-volatile main memory systems. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 310–323, 2018.

[47] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer New York, 2007.

[48] Ralph C. Merkle. Method of providing digital signatures. US4309569A, 1979. https://patents.google.com/patent/US4309569.

[49] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *Advances in Cryptology—CRYPTO '87*, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[50] Kazuhiko Minematsu and Toshiyasu Matsushima. New bounds for PMAC, TMAC, and XCBC. In *Internatinoal Conference on Fast Software Encryption (FSE)*, volume 4593 of *Lecture Notes in Computer Science*, pages 434–451. Springer, 2007.

[51] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. CacheZoom: How SGX amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems (CHES)*, pages 69–90. Springer International Publishing, 2017.

[52] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. SEVered: Subverting AMD's virtual machine encryption. In *11th European Workshop on Systems Security*, pages 1–6, 2018.

[53] Mathias Morbitzer, Sergej Proskurin, Martin Radev, Marko Dorfhuber, and Erick Quintanar Salas. SEVerity: Code injection attacks against encrypted virtual machines. In *2021 IEEE Security and Privacy Workshops (SPW)*, pages 444–455, 2021.

[54] Matheus Almeida Ogleari, Ethan L. Miller, and Jishen Zhao. Steal but no force: Efficient hardware undo+redo logging for persistent memory systems. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 336–349, 2018.

[55] Qi Pei and Seunghee Shin. Efficient split counter mode encryption for NVM. In *2021 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 93–95, 2021.

[56] Qi Pei and Seunghee Shin. Improving the heavy re-encryption overhead of split counter mode encryption for NVM. In *2021 IEEE 39th International Conference on Computer Design (ICCD)*, pages 425–432, 2021.

[57] Antoon Purnal, Lukas Giner, Daniel Gruss, and Ingrid Verbauwhede. Systematic analysis of randomization-based protected cache architectures. In *IEEE Symposium on Security and Privacy (S&P)*, pages 987–1002, 2021.

[58] Moinuddin K. Qureshi. CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping. In *51st Annual IEEE/ACM Internal Symposium on Microarchitecture (MICRO '51)*, pages 775–787, 2018.

[59] Phillip Rogaway. Authenticated-encryption with associated-data. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, CCS

'02, page 98–107, New York, NY, USA, 2002. Association for Computing Machinery.

[60] Phillip Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In Pil Joong Lee, editor, *Advances in Cryptology—ASIACRYPT 2004*, pages 16–31, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[61] Phillip Rogaway, Mihir Bellare, and John Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Trans. Inf. Syst. Secur.*, 6(3):365–403, aug 2003.

[62] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A. Joao, and Moinuddin K. Qureshi. Morphable counters: enabling compact integrity trees for low-overhead secure memories. In *51th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 416–427, 2018.

[63] Seunghee Shin, Satish Kumar Tirukkovalluri, James Tuck, and Yan Solihin. Proteus: A flexible and fast software supported hardware logging approach for NVM. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 178–190, 2017.

[64] Solid State Storage Initiative. NVDIMM Messaging and FAQ, January 2014.

[65] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 665–678, Williamsburg VA USA, March 2018. ACM.

[66] Eran Tromer, Dag Arne Osvik, and Adi Shamir. Efficient cache attacks on AES, and countermeasures. *Journal of Cryptology*, 23(1):37–71, 2010.

[67] Yukiyasu Tsunoo, Teruo Saito, Tomoyasu Suzaki, Maki Shigeri, and Hiroshi Miyauchi. Cryptanalysis of DES implemented on computers with cache. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES 2003)*, pages 62–76. Springer Berlin Heidelberg, 2003.

[68] Rei Ueno, Sumio Morioka, Noriyuki Miura, Kohei Matsuda, Makoto Nagata, Shivam Bhasin, Yves Mathieu, Tarik Graba, Jean-Luc Danger, and Naofumi Homma. High throughput/gate AES hardware architectures based on datapath compression. *IEEE Transactions on Computers*, 69(4):534–548, 2020.

[69] Thomas Unterluggauer, Mairo Werner, and Stefan Mangard. MEAS: memory encryption and authentication secure against side-channel attacks. *Journal of Cryptographic Engineering*, 9:137–158, 2019.

[70] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. PwrLeak: Exploiting power reporting interface for side-channel attacks AMD SEV. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, pages 46–66, Berlin, Heidelberg, 2023. Springer-Verlag.

[71] Xueliang Wei, Dan Feng, Wei Tong, Jingning Liu, and Liuqing Ye. MorLog: Morphable hardware logging for atomic persistence in non-volatile main memory. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 610–623, 2020.

[72] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. ScatterCache: Thwarting cache attacks via cache set randomization. In *28th USENIX Security Symposium (USENIX Security '19)*, pages 675–692, 2019.

[73] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. SEVurity: No security without integrity : Breaking integrity-free memory encryption with minimal assumptions. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1483–1496, 2020.

[74] Chenyu Yan, D. Englender, M. Prvulovic, B. Rogers, and Yan Solihin. Improving cost, performance, and security of memory encryption and authentication. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 179–190, June 2006. ISSN: 1063-6897.

[75] Fan Yang, Youmin Chen, Haiyu Mao, Youyou Lu, and Jiwu Shu. ShieldNVM: An efficient and fast recoverable system for secure non-volatile memory. *ACM Transactions on Storage*, 16(2):1–31, June 2020.

[76] Fan Yang, Youyou Lu, Youmin Chen, Haiyu Mao, and Jiwu Shu. No compromises: Secure NVM with crash consistency, write-efficiency and high-performance. In *56th Annual Design Automation Conference (DAC '19)*, number 31, pages 1–6. IEEE, 2019.

[77] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium*, 2014.

[78] Chencheng Ye, Yuanchao Xu, Xipeng Shen, Yan Sha, Xiaofei Liao, Hai Jin, and Yan Solihin. Reconciling selective logging and hardware persistent memory transaction. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 664–676, 2023.

[79] Mao Ye, Clayton Hughes, and Amro Awad. Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 403–415, Oct 2018.

[80] Zhan Zhang, Jianhui Yue, Xiaofei Liao, and Hai Jin. Efficient hardware redo logging for secure persistent memory. In *2021 IEEE 23rd Int Conf on High Performance Computing Communications; 7th Int Conf on Data Science Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud Big Data Systems Application (HPCC/DSS/SmartCity/DependSys)*, pages 41–48, 2021.

[81] Jian Zhou, Amro Awad, and Jun Wang. Lelantus: fine-granularity copy-on-write operations for secure non-volatile memories. In *ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA '20)*, pages 597–607, 2020.

[82] Kazi Abu Zubair and Amro Awad. Anubis: ultra-low overhead and recovery time for secure non-volatile memories. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 157–168, Phoenix Arizona, June 2019. ACM.

[83] Kazi Abu Zubair, Sudhanva Gurumurthi, Vilas Sridharan, and Amro Awad. Soteria: Towards resilient integrity-protected and encrypted non-volatile memories. In *54th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 1214–1226, 2021.

[84] Pengfei Zuo, Yu Hua, and Yuan Xie. SuperMem: Enabling application-transparent secure persistent memory with low overheads. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 479–492, Columbus OH USA, October 2019. ACM.