SecNVM: An Efficient and Write-Friendly Metadata Crash Consistency Scheme for Secure NVM

MENGYA LEI, FAN LI, FANG WANG, DAN FENG, XIAOMIN ZOU, and RENZHI XIAO, Huazhong University of Science and Technology

Data security is an indispensable part of non-volatile memory (NVM) systems. However, implementing data security efficiently on NVM is challenging, since we have to guarantee the consistency of user data and the related security metadata. Existing consistency schemes ignore the recoverability of the SGX style integrity tree (SIT) and the access correlation between metadata blocks, thereby generating unnecessary NVM write traffic. In this article, we propose SecNVM, an efficient and write-friendly metadata crash consistency scheme for secure NVM. SecNVM utilizes the observation that for a lazily updated SIT, the lost tree nodes after a crash can be recovered by the corresponding child nodes in NVM. It reduces the SIT persistency overhead through a restrained write-back metadata cache and exploits the SIT inter-layer dependency for recovery. Next, leveraging the strong access correlation between the counter and DMAC, SecNVM improves the efficiency of security metadata access through a novel collaborative counter-DMAC scheme. In addition, it adopts a lightweight address tracker to reduce the cost of address tracking for fast recovery. Experiments show that compared to the state-of-the-art schemes, SecNVM improves the performance and decreases write traffic a lot, and achieves an acceptable recovery time.

CCS Concepts: • **Hardware → Emerging technologies**; • **Security and privacy → Hardware security implementation**; *Database and storage security;*

Additional Key Words and Phrases: Non-volatile memory, consistency, security, metadata

Extension of a Conference Paper: This article is extended from "An Efficient Persistency and Recovery Mechanism for SGX-Style Integrity Tree in Secure NVM" [20] published in *Proceedings of Design, Automation, and Test in Europe Conference and Exhibition* (DATE), 2020. In the previous version presented in DATE, we only considered the crash recovery of SIT. In this article, we propose SecNVM, which not only achieves SIT crash consistency but also adds two new designs. On the one hand, based on the access correlation, SecNVM proposes a novel collaborative counter-DMAC scheme to avoid unnecessary memory access and persistency. On the other hand, utilizing the ADR feature and the wasted NVM bandwidth, SecNVM proposes a lightweight address tracker to reduce the write traffic of address tracking without influencing the recovery time when rebooting. In addition, we give a recovery algorithm for the improved system. In addition, we use the persistent workloads that are more in line with the characteristics of NVM instead of SPEC2006 workloads for evaluation. We also add the analysis of metadata write traffic composition, the hardware overhead, and the comparison of recovery time.

This work was supported in part by National Key R&D Program of China No. 2018YFB1003305, NSFC No. 61832020, No. 61821003, and No. 61772216.

Authors' address: M. Lei, F. Li, F. Wang (corresponding author), D. Feng, X. Zou, and R. Xiao, Huazhong University of Science and Technology, Wuhan National Laboratory for Optoelectronics, Key Laboratory of Information Storage System, Engineering Research Center of Data Storage Systems and Technology, Ministry of Education of China, Luoyu Road 1037, Wuhan, Hubei, China, 430074; emails: {lmy_up, lfhust, wangfang, dfeng, xiaominzou, rzxiao}@hust.edu.cn.

1544-3566/2021/12-ART8 $15.00
https://doi.org/10.1145/3488724

ACM Reference format:
Mengya Lei, Fan Li, Fang Wang, Dan Feng, Xiaomin Zou, and Renzhi Xiao. 2021. SecNVM: An Efficient and Write-Friendly Metadata Crash Consistency Scheme for Secure NVM. *ACM Trans. Arch. Code Optim.* 19, 1, Article 8 (December 2021), 26 pages.
https://doi.org/10.1145/3488724

1 INTRODUCTION

Non-volatile memories (NVMs) are considered as strong contenders for DRAM, owing to their high density, DRAM-like performance, and non-volatility [17, 22]. However, they also suffer from limited endurance and low write performance [24, 47]. Meanwhile, due to the non-volatile property, an attacker can easily get the contents in them by physically stealing [46]. Thus, necessary security mechanisms and metadata must be introduced to protect user data. In addition, after a sudden power failure, we have to recover the metadata into the latest and consistent state, namely metadata crash consistency, as otherwise the soundness of the system will be impaired [9, 25].

A secure NVM system needs to solve the problem of data confidentiality and integrity, which are usually implemented by **counter mode encryption (CME)** [23], **data message authentication code (DMAC)** [41], and integrity tree verification [13]. CME is used to ensure data confidentiality and prevent secret data in NVM from being directly obtained by attackers. In CME, each data cacheline has a corresponding counter, which is used to encrypt/decrypt data blocks when a data cacheline is written to or read from NVM. DMAC is applied to prevent splicing attacks and spoofing attacks. Each data cacheline has its unique DMAC. The integrity tree is employed to authenticate replay attacks. The advanced integrity trees are constructed on all encryption counters in NVM, represented by parallelizable **SGX style integrity tree (SIT)** [11, 40] and non-parallelizable **Bonsai Merkle tree (BMT)** [29, 39]. The encryption counters, DMACs, and integrity tree nodes are three critical types of metadata in the secure NVM.

However, it is not easy to implement a secure NVM system, because apart from achieving data security, it is also necessary to consider the issue of security metadata crash consistency [20, 25, 44, 49]. To improve the performance of secure NVM, volatile metadata caches are usually added in the memory controller to buffer frequently accessed metadata blocks [13]. The counter hits in the metadata cache can parallel the decryption process and memory read while DMAC and tree node hits will speed up verification. But the gap between volatile metadata cache and NVM also incurs metadata crash consistency issues in secure NVM [25, 36, 42, 45]. Specifically, when the system suffers a sudden power failure, the cached metadata will be lost. During the recovery process, the data and associated metadata in NVM may not match, causing decryption and verification to fail. A straightforward way to ensure metadata crash consistency is **strict persistency (SP)** [25]. It uses **write-through (WT)** metadata caches so that data and all related metadata are atomically written back to NVM. However, the extra NVM write traffic and performance overhead introduced by the SP scheme is unacceptable.

Although existing works have explored the problem of security metadata crash consistency [20, 25, 43, 44, 49], the secure NVM system still faces the following challenges. First, the SIT consistency mechanisms have high write traffic. Due to the special inter-layer dependency of SIT, it is impossible to restore the entire tree after a crash only counting on the recovery of leaf counters like BMT [6]. Therefore, existing works [2, 6, 16, 49] believe that SIT is an unrecoverable structure, and its updates need to be immediately persisted to NVM, which leads to unnecessary write traffic overhead. Second, some schemes [13, 43] store the counter and DMAC separately. They ignore the access correlation between the counter and DMAC in secure NVM, which results in unnecessary read, update, and persistency overhead. Third, the present metadata crash consistency mechanisms

[20, 49] use address tracking to accelerate recovery. Although they reduce the overall overhead of metadata persistency and system recovery time, the address tracking method itself accounts for a large proportion of metadata write.

To overcome the preceding challenges, we propose a novel hardware solution—SecNVM. SecNVM is an efficient and write-friendly crash consistency scheme for security metadata in NVM. SecNVM first achieves the SIT crash consistency based on a key insight: when the SIT is lazily updated [40], the latest nonce of any SIT node in the metadata cache matches the relevant child node values in NVM, providing us the possibility of restoring SIT nodes after a crash. It chooses a restrained **write-back (WB)** metadata cache to buffer the lazily updated SIT nodes, thereby reducing the NVM writes of metadata persistency. Then, it reuses the relationship between the lost nodes and the child nodes for recovery after crashes. In addition, exploiting the strong access correlation between the counter and DMAC blocks, SecNVM proposes a novel collaborative counter-DMAC metadata layout to avoid unnecessary memory access and persistency, which further improves the system performance. Moreover, utilizing the **asynchronous DRAM self-refresh (ADR)** feature and the wasted NVM bandwidth, SecNVM adopts a lightweight address tracker for fast recovery by delaying and absorbing write requests, which reduces the overhead of recovery acceleration without influencing the system recovery time. In summary, we have the following contributions:

- *Key observation and efficient crash consistency for SIT.* We find that when the SIT is lazily updated, the lost nodes after a system crash can be restored by the child nodes in NVM. Based on this observation, we propose a relaxed persistency method and achieve fast recovery for SIT after crashes, thereby guaranteeing the SIT crash consistency with low write traffic.
- *Counter-DMAC coordinated storage to reduce memory access.* According to the access correlation between metadata, we rearrange the counter and DMAC in secure NVM to avoid unnecessary memory access and persistency overhead.
- *Lightweight address tracker by write absorbing.* Utilizing the ADR feature and the available NVM bandwidth, we introduce a lightweight address tracker to reduce the write traffic of address tracking without influencing the recovery time when rebooting.
- *System implementation and evaluation.* We implement SecNVM on GEM5 [5], and experiments show that SecNVM decreases write traffic by 48.0%/39.8%/31.4% and improves system performance by 16.9%/12.5%/10.7% compared to ASIT [49]/STAR [16]/Phoenix [2] with acceptable recovery time.

2 BACKGROUND

2.1 Threat Model

In this article, we use an attack model similar to other state-of-the-art works [4, 6, 49] on hardware-based secure memory. Our trusted computing base (TCB) consists of the processor and other core parts of the operating system (e.g., security kernels). Any off-chip resource is considered unsafe, mainly including processor-memory bus and external memory. An attacker can steal secret values by snooping the bus, scanning NVM DIMM chips, or tampering with the data. We provide three types of security protection: (1) encrypt user data to protect data confidentiality; (2) authenticate user data to prevent data tampering caused by splicing attacks and spoofing attacks; and (3) use the integrity trees to resist replay attacks.

2.2 Counter Mode Encryption

The confidentiality of memory is guaranteed by data encryption on the CPU side. CME [3, 37, 46] is widely used in the state-of-the-art secure processors (e.g., Intel Xeon Processor E-family) due to its low decryption latency and high security. The encryption algorithm of CME uses a counter, a

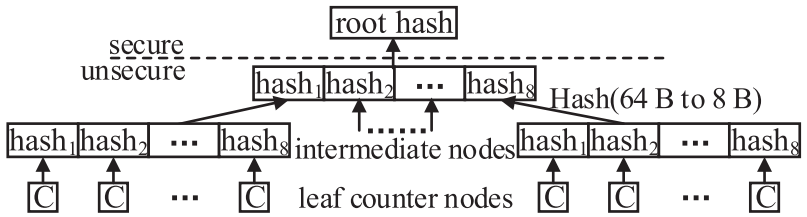

Fig. 1. The layout of an 8-ary BMT. The update process of BMT is serial.

secret key, and the data cacheline address as its input to generate a **one-time pad (OTP)** [34, 51]. When a data cacheline is written to or read from memory, it is XORed with the OTP for encryption or decryption. Each data cacheline has a unique encryption counter. The security of CME is based on the premise that the OTP related to a data block is never reused. Therefore, each time the same data cacheline is written back, the associated counter is incremented by one to ensure the temporal uniqueness of the OTP. Meanwhile, data stored in different cachelines will be encrypted by different OTPs because of the spatial uniqueness of the OTP.

2.3 Data Message Authentication Code

The encrypted data are still vulnerable to some active data tampering attacks [13]. To prevent this, the system creates a DMAC for each data cacheline. DMAC is the cryptographic signature of a data block. Each DMAC is generated from the encryption counter, the data address together with the ciphertext, using a cryptographic hash function (e.g., 64-bit AES-GCM-based GMAC [27, 41]). A DMAC is usually 64 bits long [40]. For each memory access, the system needs to recompute the DMAC. Any malicious modification in the counter, data, or DMAC can be found by comparing the value of the stored DMAC with the computed one.

2.4 Integrity Trees

The integrity tree is a widespread way to detect replay attacks, and the advanced integrity tree is built on all encryption counters in NVM for better performance [29]. Typically, any update of counters will be passed from the tree leaf to root. The root that represents the latest state of the system is securely stored in an on-chip persistent register. A memory read access needs to recompute the root hash and compare it against the saved one to verify integrity.

As shown in Figure 1, the non-parallelizable BMT is constructed by performing a layer-by-layer cryptographic hash calculation on leaf counter nodes. Each hash calculation transforms a larger child BMT node (512 bits) to a smaller hash value (64 bits) in the parent node, thereby forming an 8-ary tree. When a counter is updated, BMT will pass its latest status to the BMT root. The BMT update is performed serially from bottom to top, as the parent node depends on the child nodes. Thanks to its simple structure, the BMT, including all intermediate nodes and the root node, can be reconstructed just with leaf counternodes.

Figure 2 illustrates the structure of the parallelizable SGX SIT. Unlike BMT, each SIT node stores its own counters (also called *nonces*) instead of hash values [11]. An SIT node also keeps a **message authentication code (MAC)** that is calculated over the nonces in it and one nonce from the parent node using a secret hash function. (To distinguish it from DMAC, the MAC in SIT node is referred to as HMAC in the following text.) Except for SIT root, all SIT nodes have the same layout, which is composed of multiple nonces and a 64-bit HMAC. Whenever a leaf counter increases, the respective nonces in the parent nodes of that path are incremented and the HMACs in these nodes are updated. The SIT update can be performed in parallel because HMAC calculations use separate

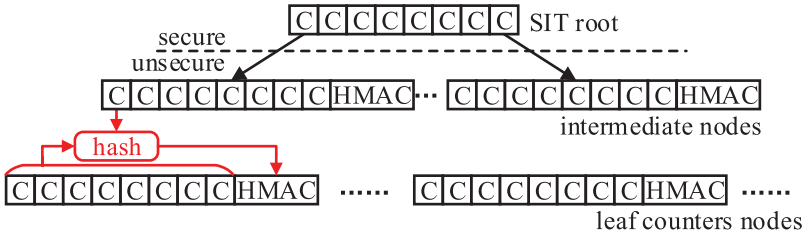

Fig. 2. The structure of an 8-ary SGX SIT. The update process of SIT is parallel.

nonces. The arity of an SIT varies with the size of the nonce. In the latest studies [31, 40], the arity of SIT reaches 128. But unlike BMT, the SIT cannot restore the intermediate nodes only through leaf counters [49].

In this work, we adopt CME and DMAC to prevent data leakage and tampering, and use SIT to detect replay attacks in NVM. Compared to BMT, SIT can be updated and verified in parallel, which improves the efficiency of integrity protection.

2.5 Metadata Crash Consistency in Secure NVM

The counter, DMAC, and SIT nodes are three main types of security metadata. Usually, they are cached in an on-chip volatile metadata cache for performance [8, 13, 32]. For example, when a data cacheline is read from NVM, we need to decrypt it using the counter. If the counter is in the cache, we can directly get the counter and perform OTP generation and data read in parallel, whereas when a counter is read from memory because of cache miss, we need to perform an integrity check. If there is no metadata cache, integrity detection from the leaf to root is required. But with a metadata cache, as long as a tree node participating in the check hits in the metadata cache, the verification is completed since the tree nodes existing in the on-chip cache are considered as secure nodes. Metadata caches greatly reduce the time of the integrity verification process and memory accesses caused by security metadata.

The metadata cache accelerates the process of data encryption and verification. But it also brings a new problem to secure NVM—that is, the metadata crash consistency [2, 6, 25, 42, 45]. Since the metadata cache is volatile, when the system suddenly crashes, the metadata in it will be lost. Even if the encrypted data block has been completely written back to NVM, the system cannot be restored correctly. This is because the latest data block and a stale encryption counter in NVM cannot be used for correctly decryption during recovery, and the stale SIT nodes will cause integrity verification errors. More seriously, the loss of the latest SIT intermediate node may cause multiple counter integrity verification errors in a large area of memory, and all the corresponding data will be discarded.

SP is a naive way to achieve metadata crash consistency in secure NVM [25]. In the SP scheme, all security metadata caches adopt the WT policy [25]. Only after the data and security metadata are atomically persisted in NVM, a data write request can be submitted. Therefore, the data and metadata in NVM will remain consistent after a system crash and the system can be restored correctly. In this case, one data write will result in multiple additional NVM writes, including writes for counter, DMAC, and SIT nodes. This solution greatly reduces system performance and NVM lifetime (see Section 5 for details). The atomic update mentioned in this work can be ensured by internal persistent registers, ADR technology [30], or log-based methods like existing works [44, 49].

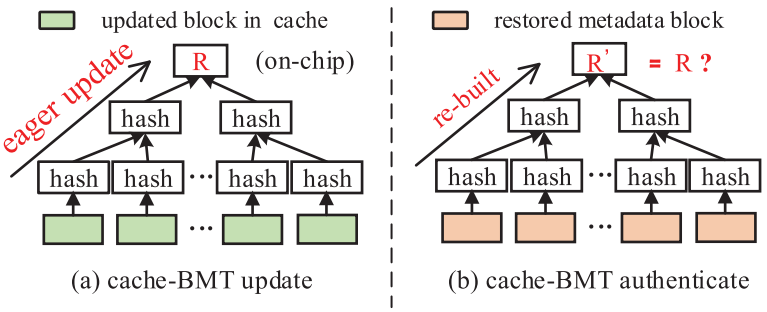

Fig. 3. The update and authentication process of cache-BMT.

2.6 Counter/DMAC Crash Recovery and Cache-BMT

Counter recovery. To cut down the overhead of counter cache with WT policy, Ye et al. [44] propose a new scheme called *Osiris* to guarantee counter crash consistency in encrypted NVM. Osiris uses a stop-loss strategy. Once a counter is updated N times, it is forcibly persisted and written back to NVM. In the common case, the counter is only updated in the cache, thus reducing the NVM writes. After a system crash, the counter stayed in NVM may be stale, and the ECC bits co-located with data are exploited to restore the latest counter. Each recovery for a counter requires at most N trials of ECC-checking, and N is compromised between runtime performance and recovery time.

DMAC recovery. To solve the crash consistency problem of DMAC, Chen et al. [6] propose the MACTree scheme according to the fixed DMAC generation formula. From the generation of DMAC, we can know that there has a fixed relationship among DMAC, data, and counter, as described in Section 2.3. After restoring the counter, we can put the counter, data, and address into the DMAC generator. Then the DMAC is restored through the re-calculation.

Cache-BMT. In a secure NVM system, an attacker may tamper with data not only during normal operation but also when the system crashes. Therefore, for the recovered security metadata after a crash, we need to check its recovery correctness. Recent studies, such as ASIT [49], CacheTree [6], and Phoenix [2], adopt a cache-BMT scheme to verify the restored metadata. The cache-BMT tracks the latest state of metadata by building a small BMT on the security metadata caches. Figure 3 shows how cache-BMT works. The leaf nodes of cache-BMT are composed of blocks in the metadata caches when the system is working normally. As shown in Figure 3(a), when a security metadata block is updated in the metadata cache, the cache-BMT is also updated accordingly at the same time. The latest state of blocks in the metadata caches is passed to the cache-BMT root node *R* through the layer-by-layer hash calculation. The root node *R* is persisted in an on-chip register. During the crash recovery process, the cache-BMT root node *R'* is rebuilt on the restored security metadata blocks like Figure 3(b). If the recalculated *R'* is equal to *R* stored on-chip, the security metadata is restored correctly. Otherwise, the restoration fails, indicating that an attack has occurred.

In this work, we guarantee the crash consistency of counters and DMACs like Osiris [44] and MACTree [6], and adopt the cache-BMT solution to verify the correctness of metadata recovery process like CacheTree [6].

3 MOTIVATION

3.1 High Persistency Overhead for SIT

It is difficult to recover SIT nodes when they are lost after a crash due to the special inter-layer dependency in SIT. The verification of each nonce in SIT nodes relies on the HMAC stored with

it and its parent node. And we cannot recover SIT from the leaf counters just by recalculation. However, if we directly adopt the SP scheme and write back all the updated SIT nodes along with the data block each time there has a data write request, the system performance will be seriously hindered. Our evaluation shows that for a 16-GB NVM system with SIT protection, the SP scheme entails significant performance degradation compared to the system without metadata crash consistency, with an average performance decline of 3.68x and additional write traffic of 8x. There are some efforts on the correct recovery of security metadata in the NVM system, but they do not consider the recovery scheme of SIT [25, 42, 45, 50]. ASIT [49] first realizes the recovery of SIT in NVM through a book-keeping mechanism that will cause more than 1x additional NVM writes. Phoenix [2] then relaxes the persistency of leaf SIT nodes, but it still has high NVM writes for intermediate nodes and addresses. STAR [16] persists the updates of an SIT node with its child nodes, but it causes extra ECC writes. This is critical for NVM, which typically has limited write endurance, higher write latency (i.e., 3–8x) and energy overhead than reads [21]. These solutions all believe that SIT is unrecoverable due to its special structure.

3.2 Inefficient Layout of Counter and DMAC

The existing counter and DMAC layouts are irrelevant. Specifically, counters are usually organized and stored in a separate counter area. For instance, in SGX, a 56-bit counter is assigned to each data cacheline, and 8 counters are packed into a counter cacheline block. Due to the limited write endurance of NVM cells (e.g., $10^7 - 10^9$ for PCM [24]), the 56-bit counters will not overflow throughout the NVM lifespan. To mitigate counter space and overflow overheads, Yan et al. [41] proposed the split counter design, which is another common organization for counters. But it needs to deal with the overflow problem of minor counters, which will result in the re-encryption of the entire page. So this article mainly considers the SGX-like 56-bit counter scheme for the general memory encryption scheme.

The general storage method of DMAC is to store DMAC in a separate metadata area, similar to the counter. In such situations, counter and DMAC will be accessed separately when serving requests, which may result in unnecessary accesses. When a read request arises, we need to read the counter to encrypt the ciphertext and read the DMAC to authenticate it, which incurs two extra memory read requests. Similarly, when serving a write request, we need to update and persist the counter and DMAC to NVM, respectively, which causes two extra NVM writes. In secure NVM, there has a strong access correlation between the counter and DMAC related to the same data block. This makes it possible for us to further improve the performance of secure NVM.

3.3 Address Tracker Overhead for Crash Recovery

Recovery time is a key factor in designing a recovery mechanism. Too long recovery time makes the recovery scheme inefficient and the persistent data fail to be recovered eventually [4]. If the entire memory is directly traversed and then the security metadata are restored one by one, the recovery time will be proportional to the NVM size. To speed up crash recovery, related works [2, 49] introduce an address tracking mechanism, which tracks the addresses of updated cache blocks and only recovers the tracked ones to accelerate recovery. To get the addresses when rebooting, it is required to persist them into NVM, which adds lots of NVM write traffic. Evaluations shown that the additional NVM write traffic caused by address tracking accounts for 50% of the total write traffic caused by metadata (see Figure 11, SecNVM_basic, for details). Therefore, how to cut down the recovery time with a smaller address tracking overhead is a key point in the crash consistency mechanism in secure NVM.

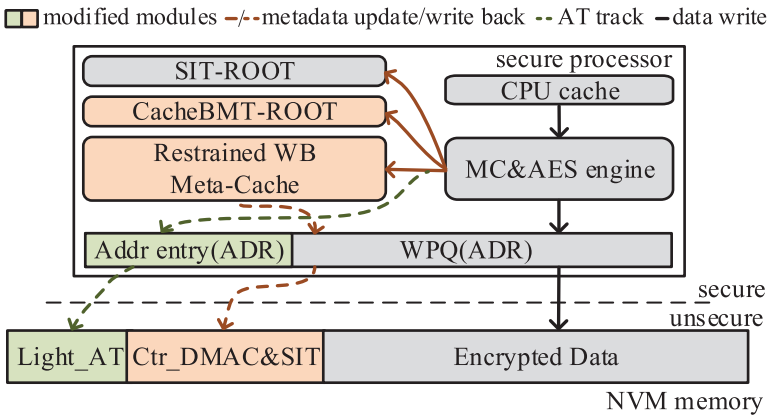

Fig. 4. The hardware architecture of SecNVM. Metadata cache write to NVM and address tracker (AT) write to ADR entry and light_AT (dotted lines) only happen under certain conditions.

3.4 Our Goal

In this article, we focus on designing an efficient secure NVM system. Its main features include the following. ① Implement the crash consistency of SIT with low NVM write traffic. We find that when the SIT is lazily updated, the lost nodes after a system crash can be restored by the child nodes in NVM. So using a more relaxed persistency scheme for SIT nodes and then recovering lost nodes after a system crash is a more friendly way to achieve SIT crash consistency. ② Utilize the strong access correlation between metadata to re-layout the counter and DMAC for high system performance. The existing layouts of counter and DMAC only consider their own locality. We aim to take advantage of the access correlation between metadata to improve the performance of metadata access. ③ Realize fast recovery with low tracking overhead. The existing address tracking mechanisms will cause a 64-B cacheline to be written back each time an address is tracked, but the effective address is only 8 B, which causes a lot of bandwidth waste. To this end, we aim to improve bandwidth utilization and reduce address tracking overhead.

4 THE DESIGN OF SECNVM

4.1 Architectural Overview

Our article proposes SecNVM, an efficient and write-friendly metadata crash consistency scheme for security metadata in NVM. SecNVM proposes a low-overhead crash recovery mechanism (Section 4.2) for SIT nodes. It first analyzes the SIT layout and finds that for the lazily updated SIT, when the system crashes, the child nodes in NVM can be used to restore the missing parent node in the metadata cache. Then, it utilizes this inherent inter-layer dependency to enable SIT crash consistency and combines it with other metadata crash recovery mechanisms. Next, leveraging the strong correlation between the counter and DMAC, SecNVM adopts a collaborative counter-DMAC optimization (Section 4.3) to improve the efficiency of security metadata access. Moreover, in Section 4.4, we propose a lightweight address tracker to reduce the tracking overhead for fast recovery. The addresses of updated metadata blocks are tracked in ADR entries and written to the address tracker in NVM when the ADR entries are full.

Figure 4 shows the overall hardware architecture of SecNVM. The added or modified components relative to the traditional secure NVM are shown in yellow and green. In addition to the CPU cache, encryption engine, write pending queue, and the NVM memory, SecNVM mainly consists of three parts: (1) a *Restrained WB Metadata Cache* that can relax the persistency of security

metadata, thereby improving the runtime performance; (2) the unified *Ctr_DMAC&SIT* area that stores collaborative counter-DMAC and SIT blocks, thereby reducing the metadata access overhead; and (3) the *Addr entry* in WPQ and *Light_AT* in NVM to accelerate the recovery process with low NVM write traffic. A cache_BMT is used to verify the recovered metadata nodes. Next, we discuss these parts in detail.

4.2 Crash Consistency for SIT

4.2.1 Key Observation for Lazily Updated SIT. When the SIT adopts a lazily updated scheme [31, 40, 49], the latest nonce of any SIT node in the metadata cache matches the corresponding child node in NVM, providing us the possibility of restoring the SIT nodes after a crash. Since the SIT nodes can be recovered after a power loss, there is no need to immediately write back the updated metadata in the metadata cache, thereby reducing the overhead caused by metadata persistency.

In the lazily updated scheme, once the data/metadata is written back to NVM from the CPU cache/metadata cache, it is sufficient to update the corresponding SIT nodes up to the first metadata cache hit. It does not immediately propagate updates to the root like the eager update scheme [14], for that any on-chip node is considered secure. This also indicates that the parent node in the metadata cache is updated only when the child node is written back to NVM. According to the structure of SIT mentioned in Section 2.4, the latest nonce in parent node stored in metadata cache and the child nodes persisted in NVM always satisfy the following equation: $HMAC\_child = hash(nonces\_child, nonce\_parent)$. Therefore, as long as there is no tampering with NVM, the lost nonces in metadata cache can be restored by the child nodes stored in NVM after a crash (the leaf counters can be recovered by ECC [44]). The HMACs in tree nodes can be recalculated after restoring all nonces.

Based on the preceding observation, we propose the crash consistency scheme for SIT in SecNVM. The main idea behind it is to minimize the overhead of SIT persistency on normal read and write, and exploit SIT's inter-layer dependency to achieve recovery after a system crash. The specific persistency and recovery processes of SIT are introduced in the next section. In addition, it is assumed that the nonces and HMAC within an SIT node are always atomically written to NVM. This can be achieved through the internal NVM registers or by providing enough backup power. Ensuring write atomicity is beyond the scope of this work.

4.2.2 Relaxed Persistency for SIT. The SP scheme realizes the crash consistency of secure NVM with SIT by a WT metadata cache. However, the WT mode enforces both data and corresponding SIT nodes to be written atomically to NVM, which greatly decreases the write performance of the system. Different from SP, SecNVM uses a restrained WB metadata cache.

Like the traditional metadata cache in secure NVM, the restrained WB metadata cache in SecNVM is mainly used to cache part of the security metadata on the processor chip, thereby speeding up the encryption/decryption and verification process. However, the restrained WB metadata cache in SecNVM adopts a new cache eviction strategy. Compared with the traditional WB metadata cache, this solution mainly adds an additional WB setting. When a nonce value in an SIT node in the metadata cache is updated to the multiple of N, we write this SIT node back to NVM. And its corresponding parent node will be updated according to the construction method of the lazily update SIT. N is named as the restrained factor in SecNVM. This special setting lays the foundation for restoring the lost SIT node after a system crash. In other cases, the metadata cache works in the normal WB mode, and SIT nodes in it are only written back to NVM when the SIT nodes are replaced and evicted. The restrained cache strategy is similar to the stop-loss strategy

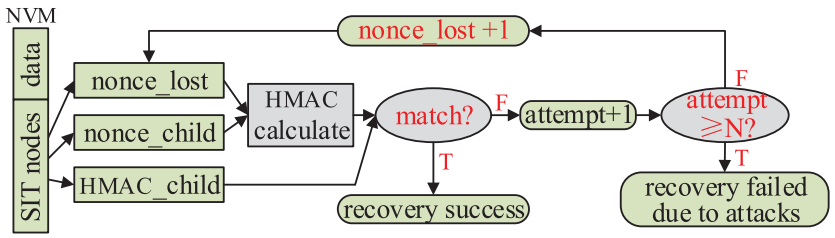

Fig. 5. SIT recovery scheme based on the inter-layer dependency of lazily updated SIT and HMAC detection. The initial value of attempt is 0.

in Osiris, but the latter is only used for encryption counters, and the recoverability of SIT has not been considered.

By the way, the metadata cache replacement strategy used in traditional secure NVM, such as first-in-first-out (FIFO) and **least-recently-used (LRU)**, are all applicable to SecNVM. Our article uses the LRU algorithm. With the restrained WB metadata cache, SecNVM aggregates several updates for an SIT node into an NVM write instead of requiring an NVM write for each metadata update. So, the restrained WB metadata cache greatly reduces the persistency overhead caused by the security metadata crash consistency mechanism.

4.2.3 Recovery for SIT. The lazy update lays the recovery foundation of SIT. For the lazily update SIT, the lost nonce of a dirty SIT node in metadata cache after a crash has the following relationship with the child SIT node stored in NVM: $HMAC_{child\_NVM} = hash(nonces_{child\_NVM}, nonce_{lost\_cache})$. So, similar to the stop-loss scheme in Osiris [44], we can bring in possible values of the lost nonce for calculation and find the nonce that satisfies this equation. In addition, the restrained WB metadata cache limits the number of possible values of the lost nonce. It will be less than or equal to the value of restrained factor (N). To ensure correct recovery ordering, when the system is initialized, SIT nodes are initialized according to zeroed counters and stored in the continuous NVM address space in the order from top to bottom and from left to right.

Therefore, we propose an SIT recovery scheme based on SIT inter-layer dependency and HMAC detection. Specifically, such as that in Figure 5, for each nonce to be recovered, we perform recovery attempts from the old value read from NVM, adding 1 to the current value each time, and trying up to N times. Find a nonce value that satisfies the following equation: $HMAC_{child\_NVM} = hash(nonces_{child\_NVM}, nonce_{lost\_cache})$. Otherwise, the recovery fails due to attacks. The HMACs in SIT nodes can be recalculated after the recovery of all nonces. And the leaf counters can be recovered by the ECC-based scheme proposed in Osiris [44]. When the system crashes, SecNVM recovers lost SIT nodes one by one with the inter-layer dependency and HMAC detection, starting from the nodes in the layer near the SIT root. See Section 4.5 for the entire recovery process of SecNVM.

4.3 Counter-DMAC Coordination

In existing secure NVM schemes, counters and DMACs are generally stored in different locations, as shown in Figure 6(a). This separate storage approach may introduce some extra accesses of security metadata. Take Figure 7(a) as an example. When a data block is read from NVM, its related counter and DMAC need to be read from the counter cache and DMAC cache, respectively. When the metadata caches are missed, two memory read requests need to be initiated to read the counter and DMAC from NVM. In addition, when a data block is written back to NVM, two metadata cache write requests are also required to update the contents of counter cache and DMAC cache. And

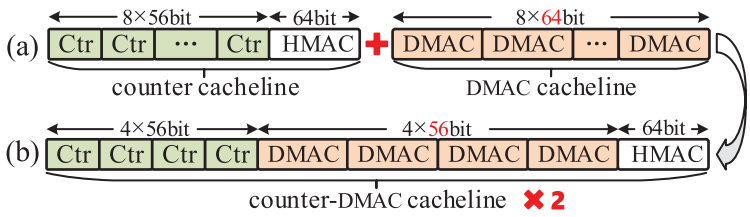

Fig. 6. (a) Counters and DMACs stored in different cachelines in traditional secure NVM. (b) A collaborative counter-DMAC storage scheme in SecNVM.

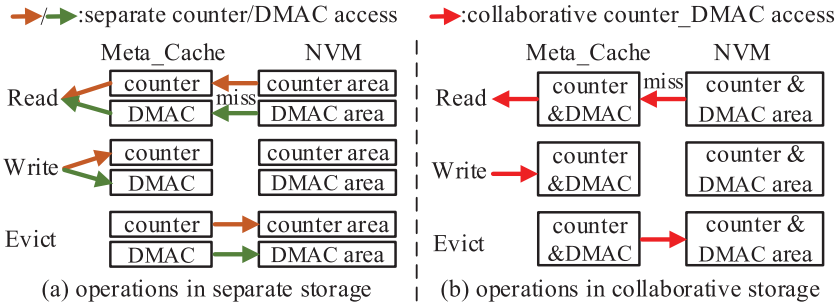

Fig. 7. Read, write, and metadata cache eviction operations in separate storage for counter and DMAC (a) and in collaborative counter-DMAC storage (b).

it will cause two NVM write operations during the eviction process of metadata cache. In short, the operations of counters are always associated with the operations of DMACs. In the separate layout of the counter and DMAC, operations such as read and write will cause two metadata cache accesses, or even two NVM accesses. Moreover, there may also exist some useless accesses in the separate layout. For example, when there are fewer than four consecutive data requests, half of the counters/DMACs in a counter/DMAC cacheline are useless accesses.

In secure NVM, a data request always results in simultaneous accesses to the corresponding counter and DMAC. Based on this strong access correlation between counter and DMAC, we design a novel collaborative counter-DMAC storage scheme to further improve the efficiency of metadata access. As shown in Figure 6(b), we store four counter-DMAC pairs and a 64-bit HMAC in the same cacheline, which serves as a child node of SIT like the original counter cacheline. The HMAC is calculated based on these four counters and a related nonce in the parent node. The counter-DMAC storage scheme adopts 56-bit DMAC instead of 64-bit. Nevertheless, 56-bit DMAC is also safe as described in Morphable Counters [31]. In the implementation, we merge the original separate NVM areas for counter and DMAC into one. On-chip counter cache and DMAC cache are also combined together.

The collaborative layout reduces counters in the SIT leaf node, which may lead to an increase in the SIT nodes and cache miss rate. To illustrate the impact, we make an evaluation of metadata cache hit rate for the separate storage and collaborative storage. The results are shown in Table 1.

As we expected, the hit rate of counter and DMAC for collaborative storage scheme is higher than that for separate storage, because the former utilizes the access correlation between counter and DMAC. Compared with the separate storage scheme, the SIT cache hit rate of the collaborative storage scheme has increased for some benchmarks (Array Swap, B-Tree, Hash Table, Queue) and decreased for other benchmarks (Red-Black Tree). This is because there are two factors that affect

Table 1. Cache Hit for Collaborative Versus Separate Counter and DMAC Storage

Counter & DMAC Hit Rate (%)						
Workload	Array Swap	B-Tree	Hash Table	Queue	Red-Black Tree	Average
Separate	87.7	93.8	87.6	96.8	87.6	90.7
Collaborative	92.7	95.3	92.5	96.8	92.4	93.9
SIT Intermediate Node Hit Rate (%)						
Separate	93.9	94.3	94.5	94.0	94.9	94.3
Collaborative	94.0	94.7	94.8	94.1	88.3	93.2
Total Metadata Hit Rate (%)						
Separate	88.6	93.8	88.5	96.6	88.6	91.2
Collaborative	92.9	95.3	92.8	96.6	91.8	93.9

the cache hit rate: factor 1 is the passive eviction strategy of the cache itself, and factor 2 is the active eviction setting of restrained WB metadata proposed in Section 4.2. This active eviction setting may cause many SIT nodes to be evicted from cache in advance, reducing the cache hit rate.

On the one hand, compared to the separate storage scheme, the number of SIT intermediate nodes in the collaborative storage scheme has increased, and the SIT hit rate will decrease (factor 1). But on the other hand, in the collaborative storage scheme, the leaf counters corresponding to an SIT node in level 1 are less, and the updates and active evictions for an SIT node in level 1 are reduced, thereby increasing the cache hit rate (factor 2). For the first four workloads, these two factors make up for each other, and the final SIT cache hit rate slightly increases. For Red-Black Tree that has relatively random access, the hit rate mainly depends on factor 1, so its hit rate has dropped by 6.6%. For all metadata caches, the overall hit rate increased by an average of 2.7%. The NVM write requests caused by passive eviction and address tracking are reduced (see Figure 11 for details).

The collaborative counter-DMAC storage greatly improves the efficiency of metadata access caused by the counter and DMAC. Figure 7(b) shows the operations of metadata under the counter-DMAC layout. Regardless of read requests, write requests, or metadata eviction, in the counter-DMAC storage, the number of accesses caused by counter and DMAC is reduced by half compared with the separate storage solution. At the same time, the collaborative counter-DMAC storage also avoids some invalid accesses compared to separate storage.

Although we use an on-chip cache to reduce the memory accesses caused by DMAC, it is also a cost-effective way to store the DMAC alongside data in the memory without cache. Using or not using DMAC cache is a trade-off between on-chip cache hardware cost and NVM bandwidth and performance overhead. For systems without DMAC cache, we can store data and DMAC in the same memory line. Then, the latency of fetching the DMAC can be overlapped with DMAC verification since the DMAC can be accessed from the same row buffer as data [19]. It also has the added advantage of consistent DMAC and data update on a write [32]. However, it incurs extra DMAC bandwidth, which is important for multi-core processors [19] and NVM with bandwidth 3 to 14x slower than DRAM [26]. For systems with DMAC cache, we can save most of DMAC bandwidth overhead, and the latency of on-chip cache is relatively shorter than that of the row buffer hit. The DMAC consistency can be promised by recovery through DMAC generation rules after a crash. However, the DMAC cache will increase the on-chip hardware cost, which can be saved for more counters or others.

The collaborative counter-DMAC storage seems to appear to overlap with ARSENAL [36]. However, they have different design goals and approaches. First, ARSENAL aims to address the high

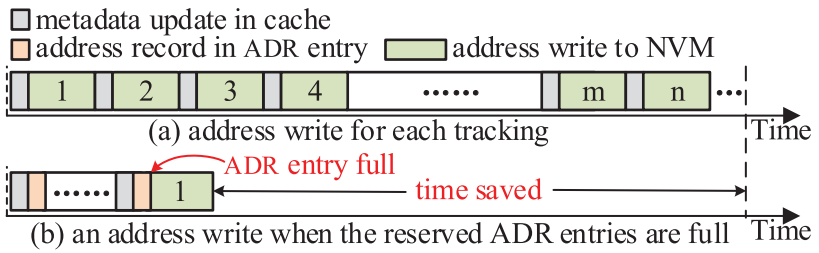

Fig. 8. Comparison of the lightweight address tracker and existing address trackers. (a) Existing address tracker writes an address cacheline for each tracking. (b) Lightweight address tracker writes an address cacheline when the reserved ADR entries are full.

overhead of atomic updates of counter, DMAC, and data, so as to support the counter/DMAC crash consistency. The collaborative counter-DMAC storage aims to improve the access performance of counter and DMAC during normal read and write requests. The counter/DMAC crash consistency is efficiently guaranteed by the recovery methods mentioned in Section 2.6. Second, ARSENAL mainly utilizes the compressibility of data and stores a counter or a DMAC or both in the redundant space of the compressed data cacheline. It tries to make the secure metadata and data persist in an atomic manner through one memory write request. But different from ARSENAL, the collaborative counter-DMAC storage exploits the strong access correlation between the counter and DMAC related to the same data cacheline. It stores four counter-DMAC pairs in the same metadata cacheline instead of the data cacheline. Third, the effectiveness of ARSENAL is related to the data compression rate. Data with a high compression rate have low metadata persistency overhead. It also needs the support of ECP bits [33] to ensure flag persistency, and it requires a separate counter layout in the system to maintain the locality of counters. These are unnecessary for the collaborative counter-DMAC storage scheme.

4.4 Lightweight Address Tracker

Address tracker. During crash recovery, due to the uncertainty of the missing security metadata, we need to traverse the entire memory for restoration. Long recovery time will increase the MTTR (Mean Time to Recovery), and may cause service unavailable. Anubis [49], Phoenix [2], and PSIT [20] recommend using an address tracker to accelerate recovery. Specifically, the address tracker persistently records the addresses of the lost metadata blocks into NVM. After a crash, the system only needs to restore the tracked ones, thus significantly reducing recovery time.

Lightweight address tracker. In the existing address tracker, each time when security metadata gets updated for the first time since its last read into the metadata cache, address tracking will be performed, thereby introducing an additional memory write. To lighten the burden, we propose a lightweight address tracker in SecNVM to delay and absorb NVM writes caused by address tracking via the ADR technology in NVM [30].

ADR is used to guarantee that the writes in the write pending queue can still be persisted into NVM via the backup battery in case of a sudden power loss. Thus, any writes reaching the write pending queue can be considered durable. We find in the previous address tracker that the address cacheline would be immediately persisted to NVM whenever an address tracking occurs. As shown in Figure 8(a), each address (8 B) will cause an entire cacheline (64 B) into NVM, resulting in a lot of waste.

Thus, we adopt a lightweight address tracker and reserve some entries (default is 1) in the write pending queue to absorb address write requests. The 1 entry is special to the address tracker and

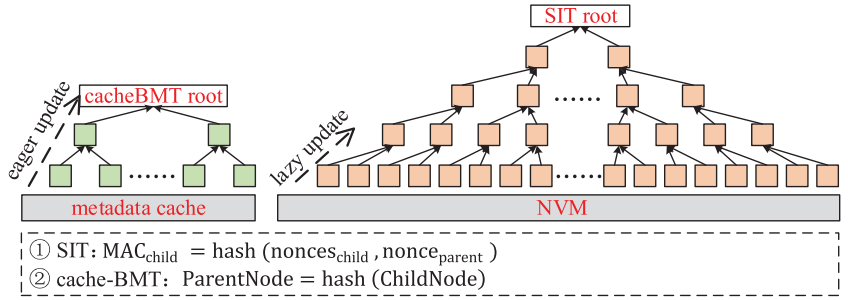

Fig. 9. Integrity trees in SecNVM. Take the 2-ary SIT and 2-ary cache-BMT as an example.

will not be persisted to NVM until they are full. The replacement of them adopts the LRU strategy. As shown in Figure 8(b), the lightweight address tracker can merge more than eight NVM writes for address cachelines into one with the ADR entry, which greatly reduces the overhead of address tracker. This is because an ADR entry can hold eight addresses, and some addresses in a entry may be invalid and replaced before the write request of this entry is triggered.

Meanwhile, for the sake of efficiency, we maintain a bitmap in the cache to indicate whether the entry of address tracker is free or not. We can easily get a free position to store the tracked address. In addition, thanks to the bitmap, when a dirty block gets evicted, we will not incur an extra write to invalidate the entry in the address tracker. On the contrary, we just need to mark this invalid state in the bitmap. In the case of power failure, the bitmap in the cache will be lost and we may try to restore some fresh metadata, but this does not matter since the correctness of the system will not be influenced.

4.5 Recovery Process of SecNVM

Recovery correctness guarantee. Similar to ASIT [49] and CacheTree [6], among others, SecNVM uses the cache-BMT scheme described in Section 2.6 to ensure the correctness of recovery. As shown in Figure 9, there are two integrity trees in the system: one is the SIT built on the entire NVM, and the other is a cache-BMT built on the dirty node in the metadata cache. For simplicity, we use a 2-ary SIT. The processing of other sizes-ary SIT is similar to it. Meanwhile, to prevent malicious tampering with the address tracker, we add the addresses to the HMAC area in cache-BMT leaf nodes. Since the HMAC in SIT nodes can be recalculated after the recovery of nonces, we do not include the HMAC when protecting the updated node by cache-BMT.

Recovery process of SecNVM. The recovery of SecNVM is divided into four steps.

① Read the addresses in the lightweight address tracker in NVM, and find the metadata blocks needed to be restored. Sort the addresses so that SecNVM starts to recover from nodes located in the layer near the SIT root.

② For an SIT intermediate node, recover the nonces separately through the inter-layer dependency and HMAC detection, and recalculate the HMAC. Specifically, the recording nodes and corresponding child nodes are read into the metadata cache. For each nonce to be recovered, we perform recovery attempts from the old value, adding 1 to the current value each time, and trying up to N times. Find a nonce value that satisfies the following equation: $HMAC\_child = hash(nonces\_child, nonce\_recovery)$, as otherwise the recovery fails. After all nonces in a node are recovered, the HMAC in this SIT node is recalculated.

③ For a leaf counter-DMAC node, recover the counters, DMACs, and HMAC. Each counter is restored through the ECC-based scheme proposed in Osiris [44], and each DMAC is recalculated and restored through its generation formula like CacheTree [6]. After the four counters in the node

are restored, the HMAC in this leaf node is recalculated on the counters and the relative nonce in the parent node.

④ Recalculate cache-BMT root, and perform an integrity check on all restored metadata blocks. The cache-BMT is reconstructed by the restored SIT nodes (including leaf counter_DMAC nodes) and node addresses after all lost nodes are recovered. If the calculated cache-BMT root is equal to the cache-BMT root stored in the on-chip persistent register, the recovery is successful. Otherwise, it means that a malicious attack occurred during the crash and the recovery failed. The pseudo-code of the recovery process is shown in Algorithm 1.

5 EVALUATION

5.1 Methodology

We use GEM5 [5], a cycle-level simulator, to evaluate the performance of SecNVM. The configuration of the target system is given in Table 2, which is similar to related works [12, 20, 25, 44, 49]. We model PCM technologies with 16-GB capacity and read/write latency of 60 ns/150 ns [18, 19]. All caches in the system adopt the LRU replacement policy. We adopt four 128-bit AES engines on-chip. For a 512-bit data block, the cycle required for encryption, DMAC generation, or HMAC computation is 40 processor cycles by default. The SIT cache in our implementation is 256 KB with 64-B cache block. For systems that adopt separate DMAC and counter storage, we simulate two 128-KB metadata caches, whereas for SecNVM with counter-DMAC layout, we adopt a combined 256-KB cache.

Workloads. We evaluate SecNVM with the benchmarks listed in Table 3, along with their memory footprint, and the number of memory read/ write requests per 1,000 instructions (RPKI/WPKI). The micro-benchmarks are similar to the benchmarks used in previous studies [25, 28, 42]. These workloads consist of persistent data structures with cacheline WB instruction (e.g., *clwb*) and memory fence instruction (e.g., *sfence*) [10].

We model the following systems for evaluation:

Write back (baseline): A secure NVM without crash consistency, which employs a WB metadata cache without any recovery methods. It is protected by separate CME, DMAC, and an eager updated SIT.

Strict persistency (SP): All security metadata caches adopt the WT policy. Once a data block is written back, the corresponding metadata are updated up to root and immediately persisted to NVM [25].

ASIT_DMAC: A variant of the advanced solution ASIT [49]. A book-keeping mechanism is used for SIT (including leaf encryption counters). For DMAC, it uses the WB cache and restores the DAMC like CacheTree [6].

Phoenix_DMAC: A variant of ASIT_DMAC [2]. It does not persist the modifications of leaf counter blocks, which can be recovered via Osiris [44].

STAR: A state-of-the-art solution for SIT crash consistency [16]. It stores the modifications of parent nodes in their child nodes. It organizes DMAC and user data in one cacheline like Synergy [32].

CacheTree: A state-of-the-art solution for crash consistency of DMAC and the non-parallelizable BMT [6]. The HNode cache size, levels, and hotness threshold are the same as recommended in the work on CacheTree.

SecNVM_basic (PSIT): A variant of the PSIT [20]. It uses the restrained WB metadata cache for all recoverable metadata. The SIT nodes are recovered by inter-layer dependency and HMAC detection as described in Section 4.2. The counters and DMACs are restored by Osiris and the fixed DMAC generation formula.

ALGORITHM 1: SecNVM Recovery

```
// 1 Read lightweight AT
Read light_AT;
sorted_AT = Sort light_AT according to address from smallest to largest;
// 2 Recover SIT intermediate nodes
for AT_i in sorted_AT and AT_i ∈ SIT intermediate address do
    SIT_i = The corresponding SIT node stored at address AT_i;
    for j ← 1 to j ← 8 do
        Read child_node_j;
        for k ← 1 to k ← N do
            if HMAC_child_j ≠ Hash(nonces_child_j, SIT_i_nonce_j) then
                recovered_SIT_i_nonce_j = SIT_i_nonce_j;
                break;
            else
                SIT_i_nonce_j++;
                k++;
            end if
        end for
        if k > N then
            The system is unrecoverable due to attacks;
            return;
        end if
    end for
    Read parent node;
    SIT_i_HMAC = Hash(SIT_i_nonces, parent_nonce);
end for
// 3 Recover SIT leaf Ctr_DMAC nodes
for AT_i in sorted_AT and AT_i ∈ SIT leaf Ctr_DMAC address do
    SIT_i = The corresponding Ctr_DMAC node stored at address AT_i;
    for j ← 1 to j ← 4 do
        Read Datablock_j;
        Fix Counter_j using Osiris[44];
        Recovered_DMAC_j = hash(key, Counter_j, Datablock_j) [6]
    end for
    Read parent node;
    SIT_i_HMAC = Hash(SIT_i_counters, parent_nonce);
end for
// 4 Verify integrity for recovered metadata nodes
Rebuild cache-BMT on recovered SIT nodes like CacheTree [6];
if rebuild cache-BMT root == cache-BMT root stored on-chip then
    The system is successfully recovered to the newest;
else
    The system is unrecoverable due to attacks;
end if
```

SecNVM_CtrD: Our SecNVM_Basic design works with the collaborative counter-DMAC optimization (Section 4.3).

SecNVM: Our solution for metadata crash consistency problem in secure NVM. Our design with all the optimizations discussed earlier (Sections 4.2, 4.3, and 4.4). On the basis of PSIT_DMAC, it

Table 2. System Configurations

Processor	
CPU	Out-of-order, X86-64, 1 GHz
L1 Cache	Private, 2 cycles, 32 KB, 2-way, 64-B block
L2 Cache	Private, 20 cycles, 512 KB, 8-way, 64-B block
L3 Cache	Shared, 32 cycles, 8 MB, 64-way, 64-B block
DDR-Based PCM Main Memory	
Capacity	16-GB PCM
Latency	Latency 60-ns read, 150-ns write [18]
Organization	2 ranks/channel, 8 banks/rank, 1-KB row buffer, Open Adaptive page policy, RoRaBaChCo address mapping
DDR Timing	tRCD 55ns, tXAW 50ns, tBURST 5 ns, tWR 150 ns, tRFC 5 ns, tCL 12.5 ns, 64-bit bus width, 1,200-MHz clock [4, 18]
Address Tracker	32 KB
Security Parameters	
En/decryption, MAC Latency	40 processor cycles [12, 49]
Counter Cache	128 KB
DMAC Cache	128 KB
Counter-DMAC Cache in SecNVM	256 KB
SIT Cache	256 KB

Table 3. Evaluated Workloads

Workload	Description	Memory Footprint	RPKI	WPKI
Array Swap	Swap random items in an array	1,006 MB	9.93	9.73
B-Tree	Insert/delete nodes in a B-tree	352 MB	4.77	4.66
Hash Table	Insert/delete entries in a hash table	1,922 MB	8.64	8.45
Queue	Insert/delete entries in a queue	2,517 MB	5.17	5.12
Red-Black Tree	Insert/delete nodes in a Red-Black Tree	870 MB	6.10	5.70

adds the collaborative counter-DMAC optimization and uses the lightweight address tracker to further reduce the write and performance overhead.

To ensure fairness, we use the restrained factor N = 8 for all schemes with restrained metadata caches unless explicitly mentioned. The average is the arithmetic mean of all workloads results.

5.2 NVM Write Traffic

Figure 10 compares the NVM write traffic incurred by different designs over the baseline system. Compared with the baseline, the SP scheme increases the write traffic by 8x on average, whereas the SecNVM_basic, SecNVM_CtrD, and SecNVM only increase 0.38, 0.22, and 0.04 times. Compared with the state-of-the-art works for SIT, SIT_DMAC, STAR, and Phoenix_DMAC, SecNVM reduces the total NVM traffic by 48.0%, 39.8%, and 31.4%, respectively. For the SP scheme, in our simulation, a 16-GB NVM requires one leaf counter block, eight internal path SIT blocks, and one DMAC block to be persisted on every data WB. This results in high NVM write traffic. Although ASIT_DMAC reduces some NVM write traffic caused by metadata through a lazily updated SIT and the DMAC WB cache, each data write will still result in an SIT shadow table write. In addition, it has some DMAC address writes for fast recovery. Phoenix_DMAC relaxes the persistency of leaf SIT nodes, but it still needs to persist every modification of SIT intermediate nodes. Moreover, each time a DMAC or SIT node is updated in cache for the first time, an NVM write caused by address tracking will be introduced in Phoenix_DMAC. STAR persists the modifications of SIT

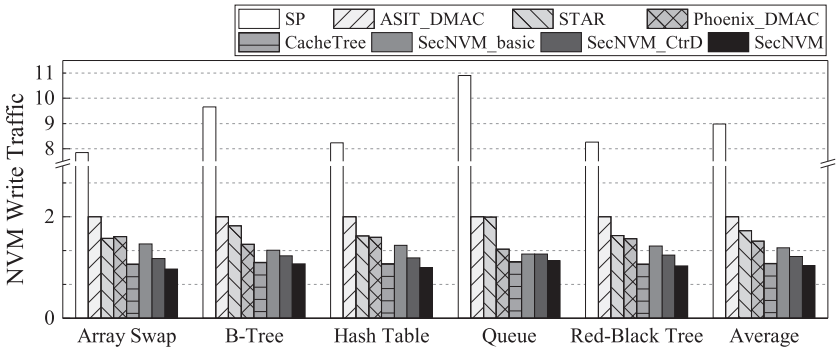

Fig. 10. NVM write traffic for different systems. Results are normalized to baseline.

parent node in its child nodes (modifications of leaf counters are stored in DMAC) and therefore eliminates the persistency write requests for SIT. It has no write requests for DMAC and DMAC address persistency, since it places the DMAC in the space usually reserved for ECC like Synergy [32]. This allows the DMAC to be fetched to the processor without requiring separate memory access. However, in this way, every data write request will be accompanied by an extra ECC write request, which cannot be ignored [6, 32]. In other schemes, ECC is stored in the same cacheline as the data, and there is no additional ECC write request.

SecNVM adopts the restrained WB metadata caches, which greatly reduces the NVM write traffic caused by both the leaf and intermediate nodes. It optimizes the write traffic of address tracking through the light_AT scheme and collaborative counter-DMAC optimization. These are very important optimizations, especially for workloads with random memory writes. For example, for hash table and array swap, the write operations are random, resulting in a large number of DMAC and SIT address tracks in Phoenix_DMAC and SecNVM_basic, making their write traffic closer to STAR, which has an extra ECC write for each data write request, whereas the address tracking overhead of SecNVM is very low due to the light_AT scheme, regardless of the workload. However, the advantage of SecNVM compared to CacheTree is not obvious in write traffic. This is because CacheTree does not perform address tracking to speed up recovery, and there is no persistency of BMT intermediate nodes. But, it has two NVM writes of two hash values when a hot BMT node is replaced from the hot tree. Compared to the baseline system, our scheme still has some extra NVM writes, especially in SecNVM_basic. We will analyze the metadata write traffic in detail in the next section.

5.3 Analysis of Metadata Write

Figure 11 demonstrates the metadata write traffic of different designs. The results have been normalized to the data write traffic. This experiment compares the composition of the metadata writes in different SecNVM systems. Overall, compared to SecNVM_basic, SecNVM_CtrD and SecNVM have reduced metadata write traffic by 30.2% and 61.4%, respectively. We have the following observations from Figure 11.

First, for three different designs, the metadata write traffic caused by the restrained WB caches (stop-loss scheme) in the counter and SIT recovery mechanisms, N_writes, are equal. Compared with the existing counter crash consistency solution, Osiris [44], there have no changes in the counter recovery methods. Over Osiris, SecNVM adds the SIT crash consistency guarantee. However, the three SecNVM designs all use the restrained WB cache, and the recovery of SIT is

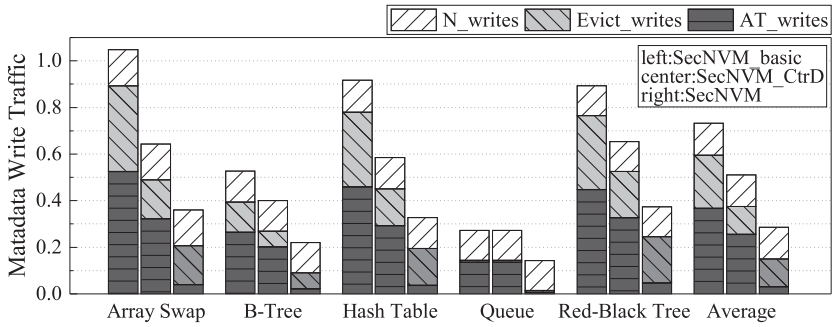

Fig. 11. Metadata write traffic for SecNVM_basic, SecNVM_CtrD, and SecNVM. Results are normalized to data write traffic.

built on the inter-layer dependencies. So, in these three schemes, N_writes has not changed. For N_writes, we can reduce it by increasing restrained factor N in the WB methods like Osiris.

Second, from SecNVM_basic to SecNVM_CtrD, the introduction of the counter-DMAC storage mechanism reduces metadata writes caused by cache eviction, Evict_writes. This is because the counter-DMAC scheme takes advantage of the strong correlation between the counter and DMAC blocks, avoiding some unnecessary accesses, thereby greatly reducing the replacement rate of the metadata cache. But for the Queue, the advantage of this optimization is not reflected due to the small working set and little evictions from the metadata cache.

Third, from SecNVM_basic to SecNVM_CtrD, the counter-DMAC storage method also reduces the write traffic caused by address tracking, AT_writes. When the counter and the DMAC are stored separately, each data WB will cause two different cachelines to be updated, and address tracking needs to write two addresses to NVM. In counter-DMAC storage, only one address needs to be written back.

Fourth, from SecNVM_CtrD to SecNVM, the use of lightweight address tracker further reduces AT_writes. The lightweight address tracker makes full use of the ADR feature and the bandwidth in NVM, fusing more than eight address writes into one.

5.4 System Performance

Figure 12 illustrates the performance of SecNVM compared to other schemes. We can observe that the SP scheme greatly extends the system execution time, which is 3.68x the baseline system that does not guarantee metadata crash consistency. ASIT_DAMC optimizes the update and persistency of SIT, and its execution time is only 0.44x that of the SP scheme, but it is still 0.62x higher than the baseline. Phoenix_DMAC further reduces the counter persistency overhead, and it has a 0.50x performance overhead compared to the baseline. The reasons for the performance degradation of ASIT_DMAC and phoenix_DMAC include the persistency of SIT, serial update of cache-BMT, and address tracking of DMAC. Compared to the baseline, STAR has 0.53x performance overhead, which is caused by the ECC writes and cache-BMT updates and reordering. The SecNVM_basic system that uses the restrained WB caches for SIT blocks performs well and only increases the execution time by 0.48x compared to the baseline system. Based on SecNVM_basic, SecNVM_CtrD further reduces the execution time through the cooperative counter-DMAC storage. The effectiveness of counter-DMAC storage is related to the working set of workloads. When the working set is large, such as array-swap, the optimization of counter-DMAC storage is obvious. This is because as the working set increases, the contention of the metadata caches becomes more serious, whereas the counter-DMAC storage effectively improves the hit rate of metadata

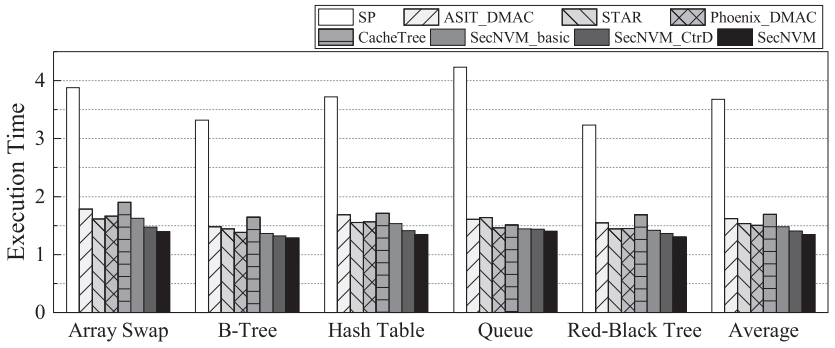

Fig. 12. System execution time for different systems. Results are normalized to baseline.

caches. On top of SecNVM_CtrD, the SecNVM scheme uses the lightweight address tracker to delay and merge the address written back, which has an average performance improvement of 4.2% over the SecNVM_CtrD design. Compared to the state-of-the-art solutions, ASIT_DMAC, SATR, and Phoenix_DMAC, SecNVM improves performance by 16.9%, 12.5%, and 10.7%. The performance overhead of SecNVM is mainly caused by the serial update of the cache-BMT. For all SIT-protected solutions, the system execution time trends are basically the same as the write traffic in Figure 10. But for the BMT-protected solution, CacheTree, the system execution time is relatively high, although it has low NVM traffic. The reason is that BMT is eagerly updated in serial but SIT is lazily updated in parallel, and the BMT update overhead is on the critical path of the system execution. Although the CacheTree solution proposes a hot tree design to alleviate this problem, the effectiveness of the hot tree is closely related to the locality of workloads, the threshold setting of the hot tree, and so on. In addition, it is worth mentioning that as the capacity of NVM grows larger, the advantages of SIT over BMT will become more obvious as the tree level increases.

5.5 Recovery Time

The metadata recovery time in SecNVM mainly includes four parts: the recovery of SIT intermediate nodes, leaf counters, DMACs, and the verification of cache-BMT. SecNVM realizes the restoration of SIT intermediate nodes through inter-layer dependency and HMAC detection. The counters are restored by the ECC-based schemes, and the DMACs are recovered through the fixed generation formula. The recovery time of counters and SIT nodes is mainly related to the number of cachelines that need to be recovered and the maximum number of attempts (N) allowed in the restrained WB cache mechanism. For each counter or SIT cacheline that needs to be recovered, its recovery time includes reading its corresponding address, reading its old value, reading its data/child cachelines for each counter and performing recovery attempt, and recalculating the HMAC. In addition, the DMAC recovery time is related to the DMAC cache size and the DMAC recompute time. Cache-BMT verification time is composed of hash calculations during reconstruction.

Figure 13 shows the comparison of recovery time for SecNVM compared to other schemes. The address tracker limits the number of metadata blocks that need to be restored to the number of dirty blocks in the metadata cache. Therefore, for gigabyte-level or even terabyte-level NVM systems, as the NVM capacity increases, the recovery time of ASIT_DMAC, STAR, Phoenix_DMAC, and SecNVM will not get longer. However, as the capacity of NVM increases, CacheTree requires longer and longer recovery times, even several hours, because it needs to traverse the entire memory for recovery. Like ASIT_DMAC, STAR, and Phoenix_DMAC, the recovery time of SecNVM is limited by the metadata cache size. Figure 14 shows the comparison of recovery time for these

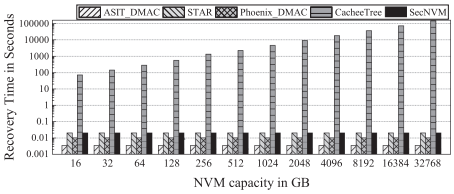

Fig. 13. Recovery time for different systems under different NVM sizes.

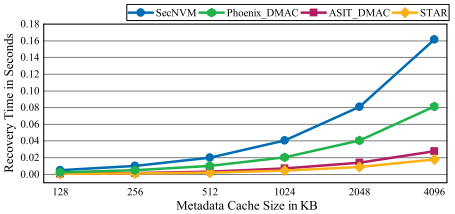

Fig. 14. Recovery time for different systems under different metadata cache sizes.

systems under different metadata cache sizes. The sizes of the counter, DMAC, and SIT cache are configured according to the ratio of 1:1:2. No matter what kind of system, the recovery time increases as the size of the metadata cache increases. This is because the metadata cache size determines the number of metadata nodes that need to be recovered. It is impractical to create a large metadata cache within the memory controller due to the chip area and lookup latency [11, 15]. So the recovery time will not be too long.

For a 4-MB metadata cache, ASIT_DMAC, STAR, and Phoenix_DMAC need 0.03, 0.02, and 0.81 seconds, respectively, to recover the stale security metadata, whereas SecNVM needs 0.16 seconds. The DMAC recovery time in ASIT_DMAC, Phoenix_DMAC, and SecNVM are the same since both of them restore the DMACs through the DMAC generation formula. The difference in recovery time between them mainly depends on the recovery of SIT nodes. In SecNVM, multiple recovery attempts are required for the restoration of an SIT node (both leaf and intermediate nodes). Phoenix_DMAC only needs to do recovery attempts for leaf nodes. However, ASIT_DMAC only needs to read the old and the new values of the lost SIT node and concatenate these two values for recovery. As for STAR, it needs not recovery DMAC since it persists DMAC with data. But it needs more NVM reads to recovery SIT nodes compared to ASIT_DMAC. Although SecNVM needs about 5.3x, 8.4x, and 2.0x recovery time than ASIT_DMAC, STAR, and Phoenix_DMAC to recover a sufficiently big (4-MB) cache for the security metadata, SecNVM also requires less than 0.2 seconds. Relative to the reduction in write overhead, the recovery time cost of SecNVM is worthy. Like Osiris, SecNVM can accelerate recovery attempts by adding AES engines. Moreover, since the systems need 10 to 100 seconds to execute a self-test after system crashes [1, 16], the 0.2 seconds for recovering security metadata in SecNVM is negligible in real-world systems.

5.6 Sensitivity to ADR WPQ Entry

The number of ADR WPQ used for lightweight address tracking (light_AT) mainly depends on the trade-off between performance and WPQ entry hardware overhead. The more entries used for light_AT, the lower the overhead of address tracking. To select the appropriate number of WPQ entries, we added a sensitivity test to the number of WPQ entries. The results are shown in Figure 15. Compared to no ADR address entry, light_AT with 1 ADR entry reduces the system execution time by 4.22%, whereas the light_AT with 16 ADR entries reduces the system execution time by 4.26% on average. For NVM write requests caused by address tracking, light_AT with 1 ADR entry and 16 entries reduce 87.7% and 88.8%, respectively, which is important for NVM systems with expensive writes. Accordingly, we can observe that with 1 ADR entry, the write requests for address tracking have been reduced a lot, and it is of little significance to adopt more ADR entries. This is because the ADR entries optimize address writing from two aspects. One is that an ADR entry can store eight addresses, so 1 ADR entry write can reduce the eight writes caused by eight addresses in the original solution to one write. The second is that the ADR entry

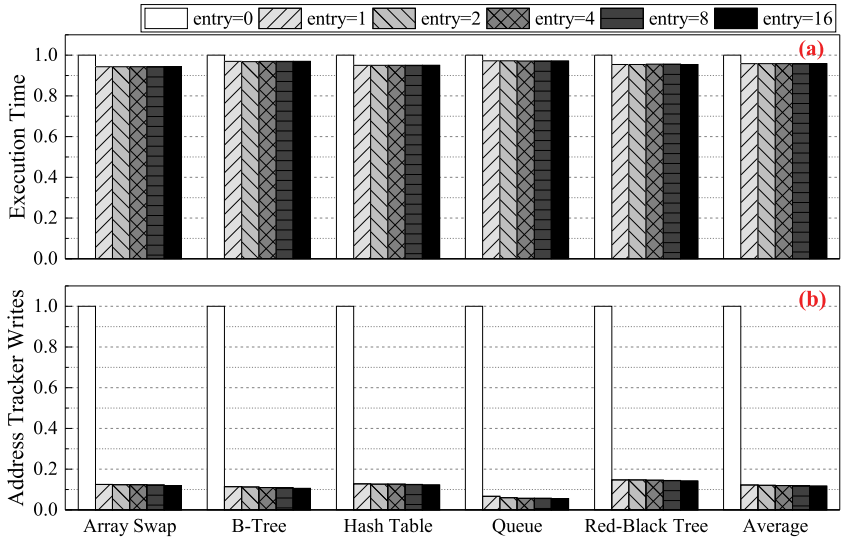

Fig. 15. System execution time (a) and NVM write traffic caused by address tracking (b) under different numbers of ADR address entries. Results are normalized to the system without ADR address entry.

can merge and absorb some address writes. Before an entry is written back, some addresses in this entry may be invalid and replaced due to the fact that related metadata blocks are replaced in metadata caches. Increasing the number of ADR entries can enhance the effect of the second aspect. However, due to the high hit rates of metadata caches (Table 1), the influence of the second aspect is not obvious. So, considering the hardware cost, we choose 1 ADR entry as the default.

5.7 Hardware Overhead Analysis

Like the regular secure NVM, we use 512-KB volatile metadata caches, a 64-B persistent SIT root register, and an encryption engine for memory encryption and integrity verification in the on-chip memory controller. And in NVM, we adopt a nearly 300-MB security metadata area for SIT nodes storage and 16 GB for data storage. These components are necessary to implement a secure NVM system, and our configuration for these components is similar to prior hardware secure memory designs without metadata crash consistency. Additionally, to achieve metadata crash consistency, SecNVM introduces some additional hardware overhead. It includes a 64-B on-chip persistent register for storing cache-BMT root, 1 ADR entries for delaying and absorbing address write requests, 16-B metadata cache area for lightweight address bitmaps that can speed up address searching and delete, and 32-KB NVM area for persisting addresses.

6 RELATED WORK

In this section, we review the relevant works in secure NVM with crash consistency. Counter-atomicity [25] first raises the issue of security metadata crash consistency. Later, ACME [35], STASH [38], Triad-NVM [4], Osiris [44, 45], CC-NVM [43], AGIT [49], EasyPM [48], and CacheTree [6] show interest in the crash consistency of counters, DMACS, and BMT nodes. Unlike these solutions, SecNVM aims at the SIT crash consistency, and adopts the Osiris and CacheTree schemes for counter and DMAC recovery. Although the restrained metadata cache strategy in SecNVM is similar to Osiris' stop-loss scheme, Osiris does not discover the special inter-layer

relationship and recoverability of SIT. It is not feasible to directly apply these solutions to SIT due to its special structure.

ASIT [49] first realizes the recovery of SIT in NVM. But ASIT concerns more with the system recovery time. It employs a book-keeping mechanism that persists the updated tree nodes immediately to achieve the recovery of SIT after a crash, which has a great impact on normal reading and writing of the system. Phoenix [2] further reduces the persistency overhead of leaf encryption counter nodes through the stop-loss scheme proposed in Osiris. SecNVM and Phoenix use the same method to ensure the crash consistency of SIT leaf nodes, but unlike Phoenix, SecNVM discovers the special inter-layer dependency of SIT and reduces the write traffic of SIT intermediate nodes. STAR [16] exploits the unused space in HMAC in one node to store the LSBs of the corresponding nonce in the parent node, thereby avoiding the additional overhead caused by the SIT crash consistency mechanism. Unlike these schemes that believe SIT is an unrecovered tree, SecNVM finds that SIT nodes can lose their latest state during normal operations and be restored through its inter-layer relationship. In addition, SecNVM proposes the collaborative counter-DMAC and lightweight address tracker schemes, and these two optimizations can be combined with existing works.

There are also some optimizations for security metadata access performance. For example, a device that realizes expedited memory protection via cache partitioning and/or data organization is proposed by Chhabra et al. [7]. With the DMAC and counter stored in the same cacheline, it accelerate memory protection operations. Lee et al. [19] adopted a shared LLC for data and security metadata to reduce the memory traffic caused by security metadata. In addition, a DM coupling scheme in which data and DMAC are co-located and a DC priority scheme in which data and counter blocks have a higher priority are also proposed, which further improves the overall performance. Unlike these works, SecNVM pays more attention to the crash consistency problems of security metadata, which is not considered in these solutions.

7 CONCLUSION

This article proposes an efficient and write-friendly solution—SecNVM—to achieve metadata crash consistency in secure NVM with high performance and fast recovery. It employs a restrained WB metadata cache and a lazily updated SIT, which reduces the write traffic caused by security metadata persistency and improves the system performance on normal operations. After system crash, the inter-layer dependency of SIT is utilized for the recovery of lost tree nodes. Then a collaborative counter-DMAC scheme is introduced to avoid unnecessary accesses and persistency overhead of security metadata. In addition, a lightweight address tracker is employed to reduce the tracking overhead. These schemes are implemented with slight modifications only on the hardware layer, which are transparent for programmers and applications. Experimental results demonstrate that compared with the state-of-the-art work, SecNVM significantly reduces the write overhead and improves system performance.

REFERENCES

[1] Wikipedia. [n.d.]. Wikipedia, Power-on self-test. https://en.wikipedia.org/wiki/Power-on_self-test.

[2] Mazen Alwadi, Kazi Zubair, David Mohaisen, and Amro Awad. 2020. Phoenix: Towards ultra-low overhead, recoverable, and persistently secure NVM. *IEEE Transactions on Dependable and Secure Computing*. Early access, August 28, 2020.

[3] Amro Awad, Pratyusa Manadhata, Stuart Haber, Yan Solihin, and William Horne. 2016. Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers. *ACM SIGPLAN Notices* 51, 4 (2016), 263–276.

[4] Amro Awad, Laurent Njilla, and Mao Ye. 2019. Triad-NVM: Persistent-security for integrity-protected and encrypted non-volatile memories. In *Proceedings of the 46th ACM/IEEE International Symposium on Computer Architecture (ISCA'19)*. ACM, New York, NY, 104–115.

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, et al. 2011. The GEM5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.

[6] Zhengguo Chen, Youtao Zhang, and Nong Xiao. 2021. CacheTree: Reducing integrity verification overhead of secure non-volatile memories. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 7 (2021), 1340–1353.

[7] Siddhartha Chhabra, Raghunandan Makaram, Jim McCormick, and Binata Bhattacharyya. 2019. Cache and Data Organization for Memory Protection. (Jan. 22 2019). US Patent 10,185,842.

[8] Siddhartha Chhabra and Yan Solihin. 2011. i-NVMM: A secure non-volatile main memory system with incremental encryption. In *Proceedings of the 38th ACM/IEEE International Symposium on Computer Architecture (ISCA'11)*. ACM, New York, NY, 177–188.

[9] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. 2011. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. *ACM SIGARCH Computer Architecture News* 39, 1 (2011), 105–118.

[10] Intel Cooperation. 2016. *Intel Architecture Instruction Set Extensions Programming Reference*. Technical Report 319433-030. Intel.

[11] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *IACR Cryptology ePrint Archive* 2016, 86 (2016), 1–118.

[12] Alexander Freij, Shougang Yuan, Huiyang Zhou, and Yan Solihin. 2020. Persist level parallelism: Streamlining integrity tree updates for secure persistent memory. In *Proceedings of the 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. 14–27. https://doi.org/10.1109/MICRO50266.2020.00015

[13] Blaise Gassend, G. Edward Suh, Dwaine Clarke, Marten Van Dijk, and Srinivas Devadas. 2003. Caches and hash trees for efficient memory integrity verification. In *Proceedings of the 9th IEEE International Symposium on High Performance Computer Architecture (HPCA'03)*. IEEE, Los Alamitos, CA, 295–306.

[14] Shay Gueron. 2016. A memory encryption engine suitable for general purpose processors. *IACR Cryptology ePrint Archive* 2016, 204 (2016), 1–14.

[15] Seokin Hong, Prashant J. Nair, Bulent Abali, Alper Buyuktosunoglu, Kyu-Hyoun Kim, and Michael B. Healy. 2018. Attaché: Towards ideal memory compression by mitigating metadata bandwidth overheads. In *Proceedings of the 51st IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. IEEE, Los Alamitos, CA, 326–338.

[16] Jianming Huang and Yu Hua. 2021. A write-friendly and fast-recovery scheme for security metadata in non-volatile memories. In *Proceedings of the 27th IEEE International Symposium on High Performance Computer Architecture (HPCA'21)*. IEEE, Los Alamitos, CA.

[17] Hoda Aghaei Khouzani, Fateme S. Hosseini, and Chengmo Yang. 2016. Segment and conflict aware page allocation and migration in DRAM-PCM hybrid main memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36, 9 (2016), 1458–1470.

[18] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. 2009. Architecting phase change memory as a scalable dram alternative. *ACM SIGARCH Computer Architecture News* 37, 3 (2009), 2–13.

[19] Junghoon Lee, Taehoon Kim, and Jaehyuk Huh. 2016. Reducing the memory bandwidth overheads of hardware security support for multi-core processors. *IEEE Transactions on Computers* 65, 11 (2016), 3384–3397. https://doi.org/10.1109/TC.2016.2538218

[20] Mengya Lei, Fang Wang, Dan Feng, Fan Li, and Jie Xu. 2020. An efficient persistency and recovery mechanism for SGX-style integrity tree in secure NVM. In *Proceedings of the 23rd Conference on Design, Automation, and Test in Europe (DATE'20)*. IEEE, Los Alamitos, CA, 702–707.

[21] Zheng Li, Fang Wang, Dan Feng, Yu Hua, Jingning Liu, and Wei Tong. 2016. MaxPB: Accelerating PCM write by maximizing the power budget utilization. *ACM Transactions on Architecture and Code Optimization* 13, 4 (Dec. 2016), Article 46, 26 pages.

[22] Zhongqi Li, Ruijin Zhou, and Tao Li. 2013. Exploring high-performance and energy proportional interface for phase change memory systems. In *Proceedings of the 19th IEEE International Symposium on High Performance Computer Architecture (HPCA'13)*. IEEE, Los Alamitos, CA, 210–221.

[23] Helger Lipmaa, Phillip Rogaway, and David Wagner. 2000. Comments to NIST concerning AES modes of operation: CTR-Mode encryption. In *Symmetric Key Block Cipher Modes of Operation Workshop, 18 Oct 2000, Baltimore, Maryland, USA*.

[24] Duo Liu, Tianzheng Wang, Yi Wang, Zili Shao, Qingfeng Zhuge, and Edwin H.-M. Sha. 2014. Application-specific wear leveling for extending lifetime of phase change memory in embedded systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33, 10 (2014), 1450–1462.

[25] Sihang Liu, Aasheesh Kolli, Jinglei Ren, and Samira Khan. 2018. Crash consistency in encrypted non-volatile main memory systems. In *Proceedings of the 24th IEEE International Symposium on High Performance Computer Architecture (HPCA'18)*. IEEE, Los Alamitos, CA, 310–323.

[26] Baotong Lu, Xiangpeng Hao, Tianzheng Wang, and Eric Lo. 2020. Dash: Scalable hashing on persistent memory. *Proceedings of the VLDB Endowment* 13, 8 (2020), 1147–1161.

[27] David McGrew and John Viega. 2004. The Galois/counter mode of operation (GCM). Submission to NIST Modes of Operation Process.

[28] Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutiu. 2015. ThyNVM: Enabling software-transparent crash consistency in persistent memory systems. In *Proceedings of the 48th IEEE/ACM International Symposium on Microarchitecture (MICRO'15)*. IEEE, Los Alamitos, CA, 672–685.

[29] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using address independent seed encryption and Bonsai Merkle trees to make secure processors OS-and performance-friendly. In *Proceedings of the 40th IEEE/ACM International Symposium on Microarchitecture (MICRO'07)*. IEEE, Los Alamitos, CA, 183–196.

[30] Andy M. Rudoff. 2016. Deprecating the pcommit instruction. https://software.intel.com/content/www/us/en/develop/blogs/deprecate-pcommit-instruction.html.

[31] Gururaj Saileshwar, Prashant Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose Joao, and Moinuddin Qureshi. 2018. Morphable counters: Enabling compact integrity trees for low-overhead secure memories. In *Proceedings of the 51st IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. IEEE, Los Alamitos, CA, 416–427.

[32] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, and Moinuddin K. Qureshi. 2018. Synergy: Rethinking secure-memory design for error-correcting memories. In *Proceedings of the 24th IEEE International Symposium on High Performance Computer Architecture (HPCA'18)*. IEEE, Los Alamitos, CA, 454–465.

[33] Stuart Schechter, Gabriel H. Loh, Karin Strauss, and Doug Burger. 2010. Use ECP, not ECC, for hard failures in resistive memories. *SIGARCH Computer Architecture News* 38, 3 (June 2010), 141–152.

[34] Shivam Swami and Kartik Mohanram. 2017. COVERT: Counter OVErflow ReducTion for efficient encryption of non-volatlle memories. In *Proceedings of the 20th Conference on Design, Automation, and Test in Europe (DATE'17)*. IEEE, Los Alamitos, CA, 906–909.

[35] Shivam Swami and Kartik Mohanram. 2018. ACME: Advanced counter mode encryption for secure non-volatile memories. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, New York, NY, Article 86, 6 pages.

[36] Shivam Swami and Kartik Mohanram. 2018. ARSENAL: Architecture for secure non-volatile memories. *IEEE Computer Architecture Letters* 17, 2 (2018), 192–196.

[37] Shivam Swami, Joydeep Rakshit, and Kartik Mohanram. 2016. SECRET: Smartly encrypted energy efficient non-volatile memories. In *Proceedings of the 53rd ACM/ESDA/IEEE Design Automation Conference (DAC'16)*. IEEE, Los Alamitos, CA, 1–6.

[38] Shivam Swami, Joydeep Rakshit, and Kartik Mohanram. 2018. STASH: Security architecture for smart hybrid memories. In *Proceedings of the 55th Annual Design Automation Conference*. ACM, New York, NY, Article 85.

[39] Meysam Taassori, Rajeev Balasubramonian, Siddhartha Chhabra, Alaa R. Alameldeen, Manjula Peddireddy, Rajat Agarwal, and Ryan Stutsman. 2020. Compact leakage-free support for integrity and reliability. In *Proceedings of the 47th ACM/IEEE International Symposium on Computer Architecture (ISCA'20)*. ACM, New York, NY, 735–748.

[40] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. 2018. VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. In *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*. ACM, New York, NY, 665–678.

[41] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. 2006. Improving cost, performance, and security of memory encryption and authentication. *ACM SIGARCH Computer Architecture News* 34, 2 (2006), 179–190.

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

[43] Fan Yang, Youyou Lu, Youmin Chen, Haiyu Mao, and Jiwu Shu. 2019. No compromises: Secure NVM with crash consistency, write-efficiency and high-performance. In *Proceedings of the 56th ACM/ESDA/IEEE Design Automation Conference (DAC'19)*. IEEE, Los Alamitos, CA, 1–6.

[44] Mao Ye, Clayton Hughes, and Amro Awad. 2018. Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories. In *Proceedings of the 51st IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. IEEE, Los Alamitos, CA, 403–415.

[45] Mao Ye, Kazi Zubair, Aziz Mohaisen, and Amro Awad. 2019. Towards low-cost mechanisms to enable restoration of encrypted non-volatile memories. *IEEE Transactions on Dependable and Secure Computing*. Early access, September 13, 2019.

[46] Vinson Young, Prashant J. Nair, and Moinuddin K. Qureshi. 2015. DEUCE: Write-efficient encryption for non-volatile memories. *ACM SIGARCH Computer Architecture News* 43, 1 (2015), 33–44.

[47] Yang Zhang, Dan Feng, Wei Tong, Yu Hua, Jingning Liu, Zhipeng Tan, Chengning Wang, Bing Wu, Zheng Li, and Gaoxiang Xu. 2018. CACF: A novel circuit architecture co-optimization framework for improving performance, reliability and energy of ReRAM-based main memory system. *ACM Transactions on Architecture and Code Optimization* 15, 2 (May 2018), Article 22, 26 pages.

[48] Zhan Zhang, Jianhui Yue, Xiaofei Liao, and Hai Jin. 2020. Efficient hardware-assisted crash consistency in encrypted persistent memory. In *Proceedings of the 23rd Conference on Design, Automation, and Test in Europe (DATE'20)*. IEEE, Los Alamitos, CA, 750–755.

[49] Kazi Abu Zubair and Amro Awad. 2019. Anubis: Ultra-low overhead and recovery time for secure non-volatile memories. In *Proceedings of the 46th ACM/IEEE International Symposium on Computer Architecture (ISCA'19)*. ACM, New York, NY, 157–168.

[50] Pengfei Zuo, Yu Hua, and Yuan Xie. 2019. SuperMem: Enabling application-transparent secure persistent memory with low overheads. In *Proceedings of the 52nd IEEE/ACM International Symposium on Microarchitecture (MICRO'19)*. IEEE, Los Alamitos, CA, 479–492.

[51] Pengfei Zuo, Yu Hua, Ming Zhao, Wen Zhou, and Yuncheng Guo. 2018. Improving the performance and endurance of encrypted non-volatile main memory through deduplicating writes. In *Proceedings of the 51st IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. IEEE, Los Alamitos, CA, 442–454.

Received April 2021; revised August 2021; accepted September 2021