An Efficient Persistency and Recovery Mechanism for SGX-style Integrity Tree in Secure NVM

Mengya Lei, Fang Wang*, Dan Feng, Fan Li, Jie Xu
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
School of Computer Science & Technology, Huazhong University of Science & Technology
Email:{lmy_up, wangfang, dfeng, lfhust, xujie_dsal}@hust.edu.cn, *Corresponding Author

Abstract—**The integrity tree is a crucial part of the secure non-volatile memory (NVM) system design. For NVM with large capacity, the SGX-style integrity tree (SIT) is practical due to its parallel updates and variable arity. However, employing SIT in secure NVM is not easy. This is because the secure metadata SIT must be strictly persisted or restored after a sudden power-loss, which unfortunately incurs unacceptable run-time overhead or recovery time. In this paper, we propose PSIT, a metadata persistency solution for SIT-protected secure NVM with high performance and fast restoration. PSIT 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 the NVM. It reduces the persistency overhead of the SIT nodes through a restrained write-back meta-cache and leverages the SIT inter-layer dependency for recovery. Experiments show that compared to ASIT, a state-of-the-art secure NVM using SIT, PSIT decreases write traffic by 47% and improves the performance by 18% on average while maintaining an acceptable recovery time.**

I. Introduction

Non-volatile memories (NVMs) are considered as strong contenders for DRAM, owing to their high density, high scalability, and low power consumption. However, NVM also faces the obstacles of limited write endurance and data vulnerability [1, 2]. To protect the NVM from malicious attacks, we must take security issues into account when designing an NVM system [3, 4].

Data confidentiality and integrity are two major aspects of NVM security, usually guaranteed by counter mode encryption (CME) and integrity tree verification [3–18]. The advanced integrity tree is constructed with all encryption counter blocks as its leaf nodes [5, 14, 17]. SGX-style integrity tree (SIT) is a practical integrity tree in secure memory [8, 9, 11, 18]. It is a parallelizable integrity tree and can tolerate greater arity than non-parallelizable integrity trees, allowing the SIT to have fewer levels. The parallelism and minor levels of SIT make it have less update and validation overhead than other integrity trees, which also reduces NVM writes.

However, it is challenging to implement SIT in secure NVM because of the problem of crash consistency. In order to improve the performance of encryption and validation, a meta-cache is usually added in memory controller to cache the frequently accessed counters and tree nodes [3, 4, 11, 12, 14–18]. Nevertheless, the volatile meta-cache also incurs crash consistency issues to secure NVM [12, 14–16]. For instance, when the system suddenly suffers a power-loss, the data is written back to NVM, but the relevant metadata may remain in the meta-cache and be lost.

Due to the inter-layer dependency limitation of SIT, it is impossible to restore the entire tree after a crash only counting on the recovery of leaf counters [18]. The loss of SIT nodes will result in the spoiled state of the tree, ruining its ability to detect replay attacks. Therefore, the encrypted data and updated metadata in the volatile meta-cache need to be written back together to NVM chips (known as secure metadata persistency) [15]. Strict persistency [15, 16, 18] is a naive approach to achieve metadata persistency of secure NVM. It requires the meta-cache to work in write-through (WT) mode. However, strictly persisting the entire tree means that a data write-back will lead to several metadata being written to NVM, which severely degrades the system performance and accelerates the NVM wear out.

When designing a secure metadata persistency mechanism for NVM, in addition to NVM write traffic and system performance, the recovery time also needs to be considered. To be more specific, in some situations, such as data centers and HPC systems, the instant data recovery (IDR) is required in the event of a system failure. In this paper, we propose **PSIT**, an efficient persistency and recovery scheme for secure metadata (including SIT leaf counters and intermediate nodes) in SIT-protected NVM. PSIT is based on a key insight: when the SIT is lazily updated, the latest nonce of any SIT node in the meta-cache matches the value of the corresponding child node in NVM, providing us the possibility of restoring SIT nodes after a crash. Based on this, we firstly choose a restrained write-back meta-cache to cache the lazily updated SIT nodes, thereby reducing the NVM writes of metadata persistency. Then, we reuse the relationship between the lost node and the child nodes for recovery after crashes. Besides, for the purpose of low recovery time, PSIT employs an address tracker on the nodes that need to be recovered. In summary, we have the following contributions in this paper:

- **Secure Metadata Persistency Scheme**. We propose PSIT, which reduces the frequency of metadata persistency through a restrained write-back (WB) meta-cache, thereby improving the system performance.
- **Recovery Method for SIT**. We achieve an effective recovery of SIT based on observation: when the SIT is lazily updated, the lost nodes after system crashes can be

restored by the child nodes in NVM.

- **System Implementation and Evaluation**. We implement PSIT on GEM5 [19] and experimental results demonstrate that PSIT decreases write traffic by 47% and improves system performance by 18% compared to ASIT[18] while maintaining an acceptable recovery time.

II. Background and Motivation

A. Threat Model

In this paper, we use an attack model similar to prior studies on secure memory [9–18]. Our trusted computing base (TCB) consists of the processor and core parts of operating system (e.g. security kernels). Any off-chip resources are considered unsafe, mainly including processor-memory bus and external memory. An attacker can attack the system by snooping the bus, scanning NVM chips or tampering with the data. Data confidentiality and integrity attacks such as splicing attacks, spoofing attacks, and replay attacks are all considered.

B. Data Encryption and Integrity Verification

Counter mode encryption (CME) is a commonly used encryption method in secure processors. In CME, the encryption algorithm uses an initialization vector that includes a counter as its input to generate a one-time pad (OTP). The security of CME needs to ensure that each OTP is never reused. When a data block is written back/read, the plaintext/ciphertext data block is XORed with the OTP to produce the ciphertext/plaintext. The recent CME scheme stores 64 minor counters together with a shared major counter in a counter block for low storage and access overhead (known as split counter [5]).

The encrypted data is still vulnerable to some active data-tampering attacks. To prevent this, the data message authentication code (DMAC) and integrity tree are employed. DMAC is the cryptographic signature of a data block. Similar to previous work [10, 15, 16], we assume that the DMAC is always written back to NVM with the data block. The integrity tree is a widespread way to detect replay attacks, and the advanced integrity tree is built on all encryption counters for better performance. Typically, any update of counters will be passed from leaf to root. The root which represents the latest state of the system is securely stored in a persistent register on-chip.

The integrity tree can be classified into parallelizable integrity trees and non-parallelizable integrity trees, represented by SGX-style integrity tree (SIT) [8, 9, 11, 18] and Bonsai Merkle Tree (BMT) [5, 12–16] respectively. SIT is widely used in secure memory due to its parallelism and lager arity. As shown in Fig. 1, all SIT nodes have the same layout, which is composed of multiple counters/nonces and a 56bit MAC. The MAC is calculated over the nonces in the node and one nonce from the parent node using a secret hash function stored on-chip. Each time a leaf counter is incremented, the respective nonces in the parent nodes of that path are incremented. Then the MACs in each node can be updated in parallel. Since SIT uses nonces to enable parallelism, the arity of SIT can vary with the size of the nonce. In the latest study [11], the arity of SIT reaches 128. Compared to BMT, SIT has more flexibility, fewer layers, and can be updated in parallel, which improves the performance of write requests. But unlike BMT, the SIT cannot restore the latest state of the intermediate nodes only through leaf counters.

Caching secure metadata in an on-chip volatile meta-cache can promote the performance of secure NVM. If a counter hits in the meta-cache, OTP generation can be performed in parallel with the memory access of encrypted data, thus removing the decryption latency from the critical read path. When an encryption counter is read from memory to CPU, we need to verify all parent nodes until the first hit in the meta-cache, since the cached tree nodes have already been verified and considered as secure. Tree node hits in meta-cache can speed up the process of integrity verification.

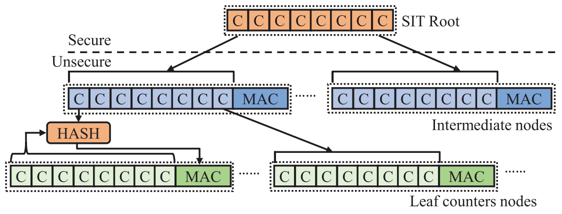

Fig. 1. SGX-style Integrity Tree.

C. Persistency and Recovery for Secure Metadata

For secure NVM, in addition to ensuring crash consistency of data, it is also necessary to ensure the persistency and recovery of secure metadata, including leaf counters and other integrity tree nodes. This is because, although the meta-cache can improve the performance of secure NVM, its volatility can lead to the loss of partial metadata after a system crash. If a counter is lost after a crash, a stale counter in NVM will cause the CME decryption to fail. More importantly, the recovery failure of integrity tree will allow attackers to exploit it. If an attacker modifies a counter to the old value during the crash, a replay attack occurs and the system cannot detect it. The most straightforward solution for this problem is strict persistency, mainly following two principles:

1) Data blocks and relevant counter blocks are written back together to NVM. This ensures the data can be decrypted correctly after the system crashes.

2) When a data block is written back, the integrity tree should be atomically updated and persisted to NVM. Otherwise, replay attacks may occur after a crash, which can damage the system security.

D. Motivation

It is difficult to recover SIT nodes when they are lost after a crash due to its special inter-layer dependency. The verification of each counter in SIT nodes relies on the MAC stored with it and its parent node. And we can't recover SIT from the leaf counters just by recalculation. However, if we directly write back all the updated SIT nodes along with the data block each time there is a write request, the system performance will be seriously hindered. Our evaluation shows that the SIT strict persistency scheme entails significant performance degradation compared to the system without metadata persistency, with an average performance decline of 95%.

More importantly, metadata persistency will bring a lot of extra NVM write requests, up to 3.9 times in our experiment. NVM promises larger capacities and higher scalability compared to DRAM. For example, the recently released Intel Optane DC persistent memory provides a large capacity up to 3TB/socket [20]. The level of an integrity tree increases as the size of the memory increases. For instance, to protect a 3TB NVM, SIT (arity=64) is up to 6 layers, which means that one data write-back will result in 6 additional NVM writes caused by SIT nodes. This is critical for NVM, which typically has limited write endurance, higher write latency (i.e. 3-8×) and energy overhead than reads [1].

We propose a novel SIT persistency and recovery scheme that greatly reduces the number of NVM metadata writes compared to strict persistency in this paper. Besides, it only introduces little overhead compared to systems without the metadata consistency guarantee.

III. PSIT

In this section, we first describe the main observation about our solution and then present the design of PSIT. Next, we introduce the operation details in PSIT. Finally, the security analysis is provided.

A. PSIT: Observation

When the SIT adopts a lazy update scheme [9, 11, 16, 18], the latest nonce of any SIT node in the meta-cache matches the corresponding child node in NVM (the leaf counters match the data and DMACs), 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 meta-cache, thereby reducing the overhead caused by metadata persistency.

In the lazy update scheme, once the data/metadata is written back to NVM from the CPU cache/meta-cache, it is sufficient to update the corresponding SIT nodes up to the first meta-cache hit. It does not immediately propagate updates to the root, for that any on-chip node is considered as secure. This also indicates that the parent node in the meta-cache is updated only when the child node is written back to NVM. According to the structure of SIT mentioned in the background, the latest nonce in parent node stored in meta-cache and the child nodes persisted in NVM always satisfy the equation: MAC_child=hash(nonces_child, nonce_parent). Therefore, as long as there is no tampering with NVM, the lost nonces in meta-cache can be restored by the child nodes stored in the NVM after a crash (the leaf counters can be recovered by the data and DMACs). The MACs in tree nodes can be recalculated after restoring all nonces. The specific recovery process of PSIT is introduced in section III-C. Besides, it is assumed that the nonces and MAC within an SIT node are always atomically written to the 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 paper.

Based on the above observation, we propose PSIT, an efficient persistency and recovery mechanism for secure metadata. The main idea behind PSIT 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.

B. PSIT: Design

Fig. 2 shows the overall architecture of PSIT. In addition to the *Encryption Engine* and *Write Pending Queue*, PSIT mainly consists of three parts: 1) *a Restrained WB Meta-Cache* which can relax the persistency of secure metadata, improving the runtime performance. 2) *An Address Tracker* in NVM to accelerate the recovery process. 3) *Mixed-Trees* to ensure the security of normal operations and recovery process. Next, we discuss these three parts in detail.

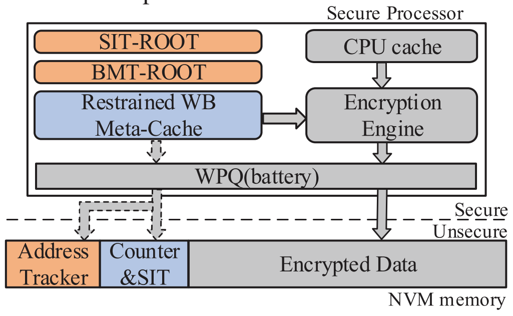

Fig. 2. Architecture of PSIT.

Restrained Write-Back Meta-Cache. The strict persistency scheme realizes the crash consistency of secure NVM with SIT by a write-through (WT) meta-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 the strict persistency, PSIT uses a restrained write-back meta-cache. When the value of an SIT node (including the leaf counters) in the meta-cache is updated to the multiple of N, we write it back to NVM. Otherwise, the meta-cache works in the normal WB mode. N is named as restrained factor in PSIT. With the restrained WB meta-cache, PSIT aggregates several updates for an SIT node into an NVM write, instead of requiring an NVM write for each metadata update. Moreover, to further reduce the overhead caused by atomic metadata update, we use the lazy update scheme described in observation for SIT. The restrained write-back meta-cache and lazy update method laid the foundation for system recovery.

Address Tracker. After a crash, the system needs to take several hours to recover, even though only the blocks in the meta-cache are lost. This is caused by the uncertainty of the missing nodes and the system will traverse the entire memory for restoration. To speed up the recovery process, PSIT persistently records the addresses of the lost nodes in the NVM address tracker.

Mixed-Trees. Due to the loss of the intermediate nodes after crashes, the SIT root cannot detect the correctness of the restored nodes. To ensure system security, we adopt a hybrid tree approach. Specifically, we establish an SIT on the entire memory to ensure the integrity of normal operations, while the integrity of the restored nodes after a crash is detected by side-BMT. Side-BMT is built on the blocks that are updated in the meta-cache but not yet persisted. The parent node is obtained by directly calculating the hash for the child nodes. The root of side-BMT (BMT-ROOT) is stored in an on-chip persistent register. Each time there is an update of SIT node in the meta-cache, the corresponding BMT nodes are updated from leaf to root. BMT-ROOT represents the latest state of the updated blocks in meta-cache. At the same time, for the purpose of preventing malicious tampering with the address tracker, we add the addresses in it to the BMT leaf nodes. Fig. 3(a) shows the structure of a side-BMT node. Since the MAC in the SIT node can be recalculated after the recovery of nonces, we do not include the MAC when protecting the updated node by side-BMT, and replace it with the node address. Fig. 3(b) shows an example of a two-layer side-BMT.

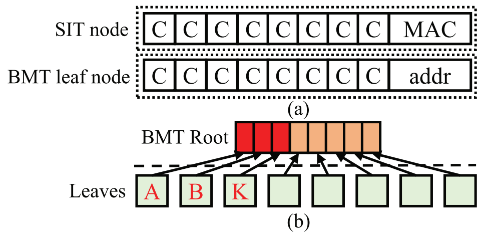

Fig. 3. (a) Organization of Nodes in Side-BMT. (b) Side-BMT for Modified Nodes and Addresses.

C. *PSIT: Operations*

The normal read process is the same as the secure NVM without a crash consistency guarantee. We show the details of PSIT mainly through the normal write and crash recovery operations.

Write. As shown in Fig. 4, during a write request, the data block is evicted from the CPU cache in step ① and the corresponding encryption counter is updated in meta-cache (if missed, the counter needs to be read from memory and verified first) in step ②. Meanwhile, if the value of the counter is updated to the multiple of N, we write it back to NVM in step ③ and go directly to step ⑥. Otherwise, we will check if it is the first update of the block since the last eviction, and if so, write back the block address to the address tracker in NVM, as shown in step ④. Similar to the write-back of data blocks, once a SIT node in the meta-cache is evicted, the parent node and the address are processed similar to step ③ and step ④. In step ⑤, we update the side-BMT according to the modified node and address until BMT-ROOT. The side-BMT nodes except BMT-ROOT can be maintained in the meta-cache. In step ⑥, the data is encrypted by the encryption engine and ⑦ written back to the NVM.

Recovery. As a result of the volatility of meta-cache, SIT nodes that have been updated in meta-cache but not persisted to NVM will be lost after a crash. Fig. 5 presents the recovery operation of PSIT. As shown by the solid line, PSIT first reads the address tracker and finds the nodes needed to be restored in step ①. According to the address information, the PSIT

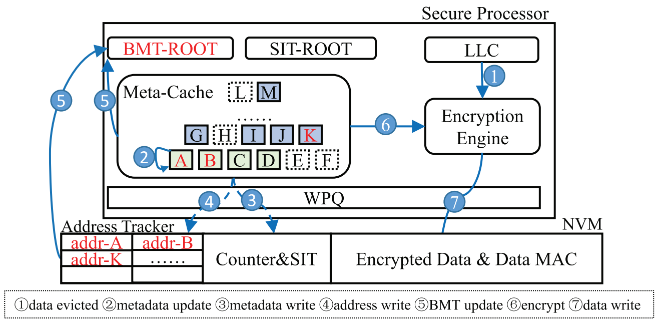

Fig. 4. PSIT Write Operation.

starts recovery from a node located in the layer close to the SIT root. In step ②, the recording nodes and corresponding child nodes are read into the meta-cache. Step ③ performs nonces recovery through inter-layer dependency and MAC detection. Specifically, 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 equation: MAC_child=hash (nonces_child, nonce_recovery), otherwise, the recovery fails. In step ④, the side-BMT is reconstructed by the nonces and addresses of the restored node after all nonces in the lost nodes are restored. The calculated BMT root is compared to the BMT-ROOT stored in the persistent register, and the recovery is considered successful when they are the same. The MAC corresponding to each node is recalculated, and the nodes in the meta-cache is updated, as shown in step ⑤. Finally, step ⑥ ⑦ and ⑧ start the normal read process. (The dotted line in Fig. 5 indicates the recovery of the leaf counters. The only difference of recovery between SIT intermediate nodes and leaf counters is that the leaf counters are restored by the data and DMACs, not the child nodes.)

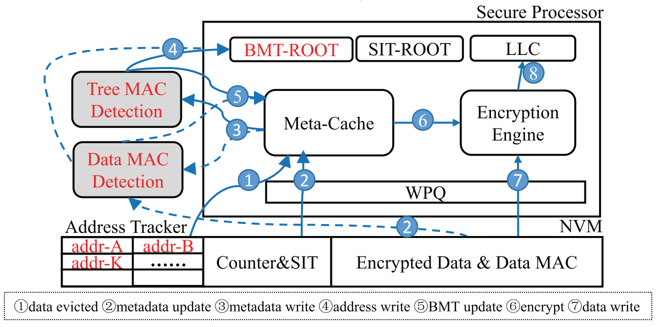

Fig. 5. PSIT Recovery Operation

D. *PSIT: Security Analysis*

PSIT has the same security level as other secure memory. During normal operation, the SIT nodes in the meta-cache always indicates the latest state of encryption counters. Any malicious attack can be found through the SIT. When the system crashes, all the restored SIT nodes and addresses in the address tracker will be verified through the side-BMT. Since the BMT-ROOT is updated with any modification of the nodes in meta-cache, it can represent the newest state of the missing nodes and detect whether there is an attack during the crash. The reuse of encryption pads will be discovered at any time of the system.

IV. Evaluation and Results

We use GEM5 [19], a cycle-level simulator to evaluate PSIT. The configuration of the target system is given in Table I, similar to related work [14, 17, 18, 21]. We use 8 memory-intensive benchmarks from the SPEC2006 suite [22]. For each benchmark, we fast forward to a representative region and then simulate 500M instructions. We model the following systems for evaluation:

Write Back (Baseline): A secure NVM without crash consistency, which employs a write-back meta-cache. It is protected by counter mode encryption and SGX-style integrity tree.

Strict Persistency (SP): Once the counter changes, the corresponding SIT nodes are updated up to root, and immediately persisted to the NVM.

AGIT Plus: A state-of-the-art metadata persistency scheme for secure NVM with BMT and split counter encryption. AGIT Plus uses a write-back meta-cache and the update is propagated to root [18].

ASIT: Secure NVM with a lazy update SIT. A book-keeping mechanism is used for SIT persistency [18].

PSIT: Our metadata persistency design for SIT-protected secure NVM with a restrained meta-cache and lazy update.

To ensure fairness, we use the arity of 64 for all SIT-protected schemes.

TABLE I
System Configurations

CPU	4-core,1GHz,X86-64,out-of-order
L1 Cache	private, 2 cycles,32KB,2-way, 64B block
L2 Cache	private, 20 cycles,512KB,8-way, 64B block
L3 Cache	shared, 32 cycles,8MB,64-way,64B block
Memory	16GB PCM,533MHz,60ns read, 150ns write
Meta-Cache	256KB,8-way,64B block
Address Tracker	32KB
En/decryption, MAC	40ns AES, 40ns MAC computation

A. System Performance

Fig. 6 illustrates the execution time of the PSIT compared to other systems with the restrained factor N=8. We can observe that for all benchmarks, the execution time of PSIT is lower than all other solutions except baseline. Compared to the baseline that does not guarantee the problem of metadata persistency, PSIT only adds an additional overhead by 1.1%, which is 18% less than ASIT on average. PSIT greatly reduces the impact of metadata persistency on system performance through the restrained WB meta-cache. For write-intensive workloads, such as libquantum, the performance improvement of PSIT will be more significant because it focuses on reducing the extra metadata writes generated by data write operations. In addition, PSIT has improved performance by 32% compared to AGIT Plus using BMT. PSIT leverages the SIT with parallel and lazy updates instead of the non-parallel integrity tree BMT, which reduces the overhead of update and verification.

In order to guarantee the security and low recovery time for secure NVM, PSIT additionally introduces the side-BMT and the address tracker. But we can easily conclude from Fig. 6 that the overhead of these methods is negligible. This is caused by 1) side-BMT is built on the updated nodes in meta-cache, and its size is related to the size of the meta-cache and benchmark locality. Since the meta-cache is much smaller than NVM, the size of side-BMT is also small. In addition, for benchmarks with high locality, the side-BMT is smaller due to the concentricity of updated nodes. 2) The address tracker is located in NVM and each time an address is updated it will result in an additional NVM write. However, in real benchmarks, the addresses of modified blocks in the meta-cache are not updated frequently.

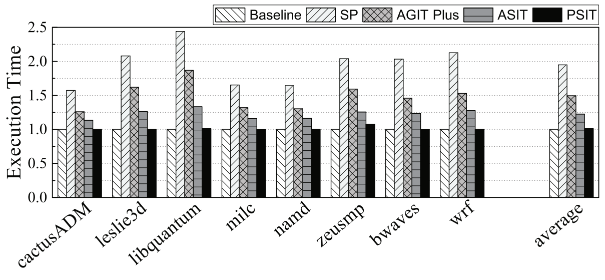

Fig. 6. System Execution Time for Different Systems. Normalized to baseline.

B. NVM Write traffic

Fig. 7 shows the impact of different designs on the number of NVM writes with N=8. Compared with baseline, the strict persistency scheme increases the write traffic by 3.9 times, while the PSIT only increases 0.05 times. In comparison to ASIT, PSIT reduces the write traffic by 47%, greatly mitigating the weakness of NVM endurance. ASIT makes a persistent copy of the meta-cache before a crash, and each update to SIT will result in an NVM write. Instead, PSIT only writes the metadata back when it is updated a certain number of times. However, the advantage of PSIT compared to AGIT Plus is not obvious in write traffic. This is mainly because AGIT Plus writes metadata back to NVM only when the meta-cache overflows. Besides, it is worth mentioning that as the capacity of NVM grows larger, the advantages of PSIT will become more apparent as the tree level increases.

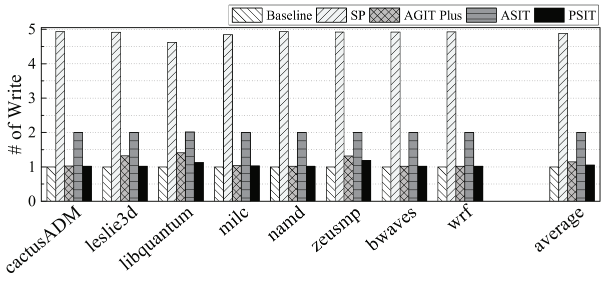

Fig. 7. Write Traffic for Different Systems. Normalized to baseline.

C. Sensitivity Study and Recovery Time

As described in Section III-B, the nonce/counter in the restrained meta-cache is written back after N updates, so the value of the restrained factor N directly affects system performance. On the other hand, since the recovery process of nonce/counter requires at most N attempts, it is also closely related to the system recovery time. Fig. 8 shows that 1) when N=8, PSIT can achieve metadata recovery in 0.11s, which is acceptable, rather than several hours like Osiris. For each SIT node that needs to be recovered, its recovery time includes: reading its address, reading the value, reading its child nodes for each nonce (counter) and performing MAC calculation to try to recover, and recalculate the MAC. For a 256KB meta-cache, we require roughly (120+64*(60+40*N)+100)*256K/64ns for recovery in the worst case in PSIT. 2) As N increases, the execution time gradually reduces for the write traffic caused by metadata decreases, whereas the system recovery time increases linearly. 3) The increase rate of performance becomes small when N is greater than 8. This is because there are few nonces that have been modified more than 8 times in the meta-cache. The optimal value of N can be selected by users based on the tradeoff between system performance and recovery time according to the benchmark characteristics.

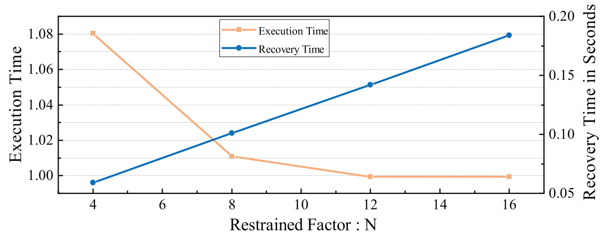

Fig. 8. Impacts with varying restrained factor N. Execution time is normalized to baseline.

V. Related Work

In this section, we review the relevant works in secure NVM with crash consistency. Counter-atomicity [12] firstly raises the issue of secure metadata persistency. It selectively persists counters according to the application characteristics to improve system performance. But it requires some complex modifications and it may cause encryption pad reuse for non-persistent memory. Osiris [14] adopts the WB counter cache to relax the persistency of encryption counters and recovers them through extra ECC bits, but it only works with encryption counters. Traid-NVM [15], CC-NVM [16] and AGIT [18] are more focused on the crash consistency of encryption counters and integrity trees in BMT-protected secure NVM. In conclusion, most existing approaches of metadata persistency are aimed at encryption counters and BMT, not at SIT. It is not feasible to directly apply these solutions on SIT nodes (including leaf counters) due to its special structure. The state-of-the-art solution, ASIT [18] first realizes the recovery of SIT in NVM. But ASIT concerns more with the system recovery time. ASIT 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. Unlike these solutions, PSIT focuses on the SIT-protected secure NVM and tries to minimize the overhead of metadata persistency while ensuring fast recovery.

VI. Conclusion

PSIT is an available crash consistency mechanism for secure NVM that uses the SGX-style integrity tree with high performance and fast recovery. It employs a restrained write-back meta-cache and a lazily updated SIT, which reduces the writes traffic caused by secure metadata and improves the system performance on normal operations. After a crash, the inter-layer dependency of SIT is utilized for the recovery of the lost tree nodes. In comparison to ASIT [18], the latest secure NVM with crash consistency, PSIT improves performance by 18% and reduces writes traffic by 47% on average, and an acceptable recovery time is achieved.

Acknowledgment

This work is supported in part by National Key R&D Program of China No.2018YFB10033005, NSFC No.61832020, No.61821003, No.61772216, National Defense Preliminary Research Project (31511010202), Hubei Province Technical Innovation Special Project (2017AAA129), Wuhan Application Basic Research Project (2017010201010103), Fundamental Research Funds for the Central Universities.

References

[1] Z. Li, R. Zhou, and L. Tao, "Exploring high-performance and energy proportional interface for phase change memory systems," in *HPCA*, 2013.
[2] S. Nalli, S. Haria, M. D. Hill, M. M. Swift, H. Volos, and K. Keeton, "An analysis of persistent memory use with whisper," in *ASPLOS*, 2017.
[3] V. Young, P. J. Nair, and M. K. Qureshi, "Deuce: Write-efficient encryption for non-volatile memories," in *ASPLOS*, 2015.
[4] S. Swami and K. Mohanram, "Covert: Counter overflow reduction for efficient encryption of non-volatlle memories," in *DATE*, 2017.
[5] B. Rogers, S. Chhabra, M. Prvulovic, and S. Yan, "Using address independent seed encryption and bonsai merkle trees to make secure processors os- and performance-friendly," in *MICRO*, 2007.
[6] B. Gassend, G. E. Suh, D. Clarke, M. V. Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *HPCA*, 2003.
[7] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and S. Yan, "Improving cost, performance, and security of memory encryption and authentication," in *MICRO*, 2006.
[8] S. Gueron, "A memory encryption engine suitable for general purpose processors," *IACR Cryptology ePrint Archive*, 2016.
[9] M. Taassori, A. Shafiee, and R. Balasubramonian, "Vault: Reducing paging overheads in sgx with efficient integrity verification structures," in *ASPLOS*, 2018.
[10] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "Synergy: Rethinking secure-memory design for error-correcting memories," in *MICRO*, 2018.
[11] G. Saileshwar, P. Nair, P. Ramrakhyani, W. Elsasser, J. Joao, and M. Qureshi, "Morphable counters: Enabling compact integrity trees for low-overhead secure memories," in *MICRO*, 2018.
[12] S. Liu, A. Kolli, J. Ren, and S. Khan, "Crash consistency in encrypted non-volatile main memory systems," in *HPCA*, 2018.
[13] P. Zuo, Y. Hua, and Y. Xie, "Supermem: Enabling application-transparent secure persistent memory with low overheads," in *MICRO*, 2019.
[14] M. Ye, C. Hughes, and A. Awad, "Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories," in *MICRO*, 2018.
[15] A. Awad, L. Njilla, and M. Ye, "Triad-nvm: Persistent-security for integrity-protected and encrypted non-volatile memories," in *ISCA*, 2019.
[16] F. Yang, Y. Lu, Y. Chen, H. Mao, and J. Shu, "No compromises: Secure nvm with crash consistency, write-efficiency and high-performance," in *DAC*, 2019.
[17] S. Liu, K. Seemakhupt, G. Pekhimenko, A. Kolli, and S. Khan, "Janus: optimizing memory and storage support for non-volatile memory systems," in *ISCA*, 2019.
[18] K. A. Zubair and A. Awad, "Anubis: ultra-low overhead and recovery time for secure non-volatile memories," in *ISCA*, 2019.
[19] N. Binkert *et al.*, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1–7, 2011.
[20] Intel, "Enhancing High-Performance Computing with Persistent Memory Technology," 2017.
[21] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable dram alternative," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 2–13, 2009.
[22] C. D. Spradling, "Spec cpu2006 benchmark tools," *SIGARCH Comput. Archit. News*, vol. 35, no. 1, pp. 130–134, 2007.