Soteria: Towards Resilient Integrity-Protected and Encrypted Non-Volatile Memories

Kazi Abu Zubair
Department of ECE
North Carolina State University
USA
kabuzub@ncsu.edu

Sudhanva Gurumurthi
RAS Architecture
Advanced Micro Devices, Inc.
USA
Sudhanva.Gurumurthi@amd.com

Vilas Sridharan
RAS Architecture
Advanced Micro Devices, Inc.
USA
Vilas.Sridharan@amd.com

Amro Awad
Department of ECE
North Carolina State University
USA
ajawad@ncsu.edu

ABSTRACT

Although emerging Non-Volatile Memories (NVMs) are expected to be adopted in future memory and storage systems, their non-volatility brings complications in designing processors wherein security is an essential requirement. One of these complications is maintaining the correctness of the security metadata for encryption and integrity verification. Due to the accommodation of security metadata in the NVMs, they are susceptible to reliability threats posed by the underlying memory technology. This is undesirable because the secure operation of the system highly depends on the correctness of the security metadata stored in the memory.

We observe that the error sensitivity of security metadata is higher than general data and requires special attention. A single uncorrectable error in a top Merkle tree node can leave a large portion of memory data unverifiable. To solve this, we propose *Soteria*, a scheme that provides higher error-tolerance of security metadata by lazily duplicating them. Soteria decouples existing memory reliability from the security metadata reliability and achieves security, performance, and high reliability within the same system with only minor memory controller changes. Our proposed scheme improves the reliability of the improved security NVM system significantly while causing only about 1% system slowdown on average.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**.

KEYWORDS

Non-Volatile Memory, Memory Security, Memory Reliability

ACM Reference Format:
Kazi Abu Zubair, Sudhanva Gurumurthi, Vilas Sridharan, and Amro Awad. 2021. Soteria: Towards Resilient Integrity-Protected and Encrypted Non-Volatile Memories. In *MICRO'21: 54th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '21), October 18–22, 2021, Virtual Event, Greece.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3466752.3480066

MICRO '21, October 18–22, 2021, Virtual Event, Greece

ACM ISBN 978-1-4503-8557-2/21/10…$15.00
https://doi.org/10.1145/3466752.3480066

1 INTRODUCTION

The long-awaited emerging Non-Volatile Memories (NVMs) are finally in the market. Intel's and Micron's 3D XPoint is a prominent example of emerging NVMs that have the potential for replacing (or augmenting) DRAM to build future memory systems[18, 43]. Unlike DRAM, emerging NVMs can retain data for a long time after power-loss, which enables applications that rely on persistence, such as checkpointing and filesystems[8, 25]. Moreover, many emerging NVMs feature a very high density and thus promise large capacities for future memory modules. Additionally, many emerging NVMs have ultra-low idle power; they do not require frequent refresh operations as in DRAM. Finally, with latencies comparable to DRAM, NVMs can achieve an acceptable level of performance when used as the main memory. On the other hand, promising NVM technologies, such as Phase-Change Memory (PCM) – suspected to be the technology behind 3D XPoint, suffer from limited write endurance and slow write operations. Moreover, the data remanence property of NVMs expands the attack surface and further facilitates physical access attacks[5, 10, 45].

Since NVMs retain data for a long time after power loss, protecting data confidentiality and integrity becomes necessary[5, 10, 27, 45, 47]. Fortunately, processor vendors are rightfully adding support for protecting the integrity and confidentiality of data through native support for integrity verification and encryption in memory controllers[12, 16, 29]. Although such support is necessary for NVMs, it is also used to protect against cold-boot attacks in DRAM and very advanced threat models (e.g., untrusted Operating System). While such memory security support is essential for NVMs, the limited write-endurance and expectancy of data recovery require special considerations on how such support is implemented[4, 5, 45, 49]. Recent work explored mechanisms to recondition the current implementations of secure memory to allow recovery[45], reduce recovery time [49], and reduce the performance overheads for enabling system recovery[5, 27].

While two of the CIA (Confidentiality, Integrity, and Availability) triad elements are relatively well understood and explored for NVMs, the availability of integrity-protected and encrypted NVMs is rarely explored. The reliability of data itself (whether encrypted or not) and how to recover from errors have been studied in recent work[48]. However, the impact of implementing integrity-protection and encryption on the overall system resilience and data availability has been rarely discussed. In particular, how errors can affect the availability of data and the ability to recover the system after a crash (or power loss) in secure memories. Surprisingly, a small number of errors in security metadata can render significant amounts of data unrecoverable (or at least unverifiable). In conventional memory systems, we can have data with different sensitivity levels. For instance, we can imagine a memory error makes its way to the kernel code or the root of the page table. Most likely, such an error will affect one or a few memory locations before the kernel detects the error or crashes. Meanwhile, an error in a user application's data may cause the termination of only that specific application while the system can continue to operate. Sensitive software-visible data and files can be duplicated or have their reliability further improved by adding software-level redundancy mechanisms and integrity checking. Unlike software-visible data, the hardware-managed security metadata is more challenging to protect for several reasons. First, security metadata is invisible to the OS or system software, and thus its reliability protection is limited only to the memory ECC. Second, errors in security metadata can directly propagate to a huge number of memory locations by rendering them unverifiable. Third, errors in security metadata leave the system with one of two options: accept a security risk and possible data tampering of the associated memory locations, or erase all potentially affected data, directly or indirectly (unverifiable data).

The most common type of error is soft errors. Soft errors occur due to internal or external interference, where the memory cell is still functional, but the stored value has been changed. In conventional (non-persistent) memory systems, once the memory controller reads a memory location with uncorrectable error(s), it triggers a system reboot. While rebooting the computer remedies soft errors in non-persistent memories, this is not the case for persistent memories. In persistent memories, the state of the memory before a crash cannot be simply discarded and boot-up in a clean slate; many applications and OSes leverage the persistence of NVMs to allow data recovery after crashes. In other words, in persistent memories, errors *persist* along with data and hence rebooting the system can no longer mask the soft errors. With that in mind, when uncorrectable errors are detected by the memory controller, the system might take further measures by alerting the OS or allowing the OS to fix the error. OS can then choose to mask the errors (e.g., set to zeros), or delegate the error correction to the application that has read the data, or even terminate the application. In Linux, a feature called *hardware poisoning* (HWPOISON)[22], leverages processor support to flag uncorrectable error data as poisoned. Once such poisoned data is being used by the application, a fault will be issued and can be handled by the application or allow the OS to kill the application. Techniques such as hardware poisoning, software-based redundancy, or simply masking (or initializing) corrupt values can be used in response to software-visible memory errors in persistent memories. However, errors in software-transparent security metadata need special handling.

Security metadata includes encryption counters and Merkle-tree and can occupy a large space in memory if the whole memory is protected. Merkle-tree is a hierarchical structure where leaves (encryption counters) are hashed in multiple levels, and finally, it converges into a single hash value (the root of the tree), which can be stored within the processor chip. While some types of Merkle Tree (e.g., BMT) allows reproducing the root merely from the leaves, it is not the case for some tweaked (and more secure) integrity trees (e.g., Tree of Counters (ToC)[16, 49]). Errors in integrity trees are particularly very challenging and can even prevent recovery in some tree structures like ToC. Depending on which level they occur on, the tree update mechanism, and integrity tree type, errors can render the system fully or partially unrecoverable or unverifiable. In current implementations of secure memory, commercial processors adopt a *drop-and-lock* policy[12, 16], where any failure of integrity verification of a data block causes the whole system to lock and reboot. Note that such processors have been designed with DRAM systems in mind. However, as mentioned earlier, in persistent memory, errors persist as well, and thus merely rebooting the system will not be able to solve the problem.

Correctable errors can be first fixed before trying to verify the integrity of data, which can also serve as a guarantee that the data has been corrected. Note that an uncorrectable error in data or metadata will trigger an integrity verification failure, and the underlying data will be marked as unverifiable. The higher in the tree an uncorrectable error occurs, the more data rendered unverifiable. Note that such a region can be hundreds of gigabytes of memory. Accepting the current contents of the region opens the room for a wide range of attacks, such as known-plaintext attacks leveraging counter replay or simply data replay attacks. Meanwhile, just reinitializing the whole affected region in memory can erase essential data, which might affect the availability of sensitive applications, or permanently delete sensitive files. Note that even reinitializing the covered memory region (including encryption counter) can potentially cause counter value replay attacks, which can be prevented by re-encrypting the whole memory with a new key, a very lengthy and expensive process that can take hours[49].

Due to this 'speciality' of security metadata residing in NVM, and the 'amplified' effect if an uncorrectable error happens in the metadata region, they demand more robust protection from errors. However, uniformly increasing the strength of the ECC just because of the added sensitivity by the security metadata is expensive and unattractive. Although having stronger ECC exclusively for the metadata region can be a solution, it is not desirable considering the design complexity, extra power consumption, and added latency to read and write metadata. Previous works show that having stronger ECC can have latency increase up to 21% [11]. Besides, memory modules are expected to be designed base on the RAS requirement for the target market, which is determined considering acceptable data reliability, not the security-introduced problems. The security implementations should be near the processor chip, which needs to be designed considering the security, persistence of metadata, crash-consistency, and minimal performance overhead. Accordingly, any reliability degradation by the security metadata must be addressed

within the security implementation, not by linearly strengthening the underlying ECC.

In this paper, we argue that even though the reliability of sensitive software-visible data can be protected through traditional fault tolerance techniques by software, hardware-managed security metadata solely relies on the reliability support in memory, and thus needs special consideration. Errors in security metadata can override and render such mechanisms ineffective. Accordingly, we advocate for novel software-transparent mechanisms to improve the reliability of security metadata. To this end, we realize that the resilience for the security metadata should be decoupled from the memory reliability to allow parallel and uninterrupted development in both sides separately. In this paper, we propose a novel secure memory controller design, Soteria, that improves the reliability of integrity-protected systems. Soteria design is based on our observation that upper levels of integrity tree cover more substantial parts of the memory, but their storage overhead is minimal compared to low levels; for an 8-ary tree, we have 4096x smaller number of nodes in Level five versus those in Level one. Accordingly, traditional fault-tolerance techniques, such as redundancy, can be used in upper levels. Moreover, Soteria also utilizes the fact that certain tree update procedures allow for higher redundancy in the upper level with the cost of negligible performance overhead. This allows Soteria to effectively limit the memory region that is affected due to errors in security metadata. In a nutshell, Soteria makes the following contributions.

- We investigate the impact of errors in security metadata and propose smart metadata management schemes to make the improved security NVM systems more resilient.
- We propose Soteria, which uses duplication method without incurring significant performance overheads.
- We keep our implementation within the security metadata management system, decoupled from the traditional system reliability, which essentially results in independent development in both RAS and security.

To evaluate the performance of our proposed schemes, we model Soteria in gem5 (publicly available open-source version), a cycle-accurate architectural simulator[7], and evaluate the performance using a set of persistent and non-persistent memory-intensive workloads from Whisper benchmarks[31], PMEMKV[1], SPEC CPU® 2006 benchmark suite[21], and a mix of in-house synthetic persistent benchmarks. Our evaluation shows that Soteria can be integrated with negligible performance overheads. We compare our scheme with state-of-the-art secure, and crash-consistent NVM implementation [49]. On average, Soteria incurs an additional execution time overhead of about 1% while maintaining the reliability of the integrity tree. Moreover, we use an industry-grade memory fault simulator FaultSim[34] to evaluate resilience and define *Unverifiable Data Ratio (UDR)* to be a metric for security metadata related resilience measurements. Our experimental evaluation shows that Soteria can keep the unverifiable data loss extremely low and demonstrates approximately 3.7×10^4 times more resilience to errors in security metadata.

The rest of the paper is organized as follows; Section 2 discusses about the background of our work and the motivation, Section 3 illustrates the design, Section 4 discusses the evaluation methodology and Section 5 analyzes the results. Finally, Section 6 discusses security and reliability of our scheme, Section 7 discusses the previous works, followed by a conclusion.

2 BACKGROUND AND MOTIVATION

In this section, we describe the concepts that are fundamental to our scheme, Soteria.

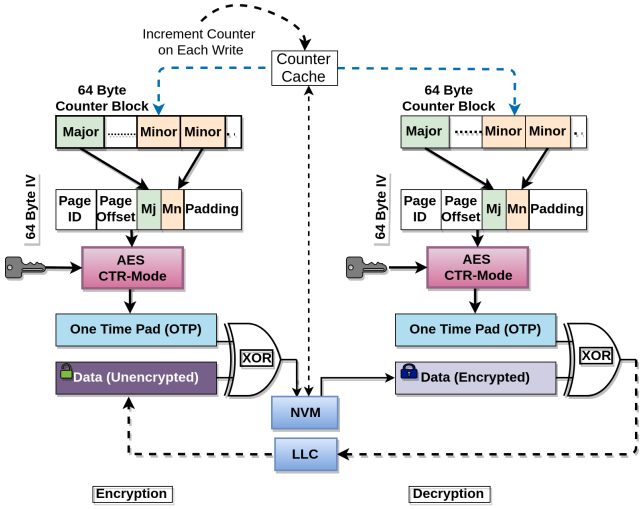

Figure 1: Counter-mode encryption and decryption (using split-counter scheme) [4, 45].

2.1 Threat Model and Assumptions

Similar to other state-of-the-art works in improved security NVM systems [4, 6, 9, 35, 45], the trust base of Soteria is only the processor chip. Any content stored outside the processor chip is considered vulnerable to external attacks. The attackers might attempt to snoop the bus, scan the memory, or replay previously captured memory blocks. In addition to that, we assume that there might be a sudden crash or power outage at any point in time that the system should effectively handle, and the memory is susceptible to typical errors common to NVM modules.

2.2 NVM Technology

NVM refers to a wide range of non-volatile technologies, including Phase Change Memory (PCM), Resistive RAM (ReRAM), Spin Transfer Torque RAM (STT-RAM), NAND Flash, all of which have a specific fault model to be considered. Although we aim not to constrain our solution to any specific technologies and their inherent fault characteristics, we note that Phase Change Memory (PCM) is a more matured technology compared to other technologies, and industry is adopting this technology for future NVM based main memory (e.g., Intel's 3dXPoint[20]). While we leave the choice of technology, fault characteristics, and memory reliability techniques as an orthogonal direction to our work, we focus on having a reliable Merkle Tree structure and metadata management scheme that can improve the reliability of an improved security NVM system regardless of what memory technology is being used.

2.3 NVM Errors and ECC Mechanism

While the cause of errors in a memory module can be specific to the memory technology being used, they can be generalized as permanent or transient errors. Transient errors in NVM can be caused by different phenomena, including resistance drift (in MLC PCM) and retention failures. Error Correcting Codes (ECC) or Error Correcting Pointers (ECP)[38] are generally used to correct errors in the memory, requiring additional chips to store ECC codes calculated over the data. Unlike DRAM, NVM-based main memories require more reliable ECC protection capable of correcting multi-bit errors. For example, in MLC PCM, once a cell loses its resistance level, other adjacent cells are highly likely to produce an incorrect state in the near future [28]. Consequently, NVM-based main memories are expected to have additional chips for allowing stronger error correction (e.g., Intel's Optane DIMM has eight chips for data and three chips for ECC). Stronger ECC, for example, BCH (Bose, Ray-Chaudhuri, Hocquenghem) code, LDPC (Low-Density Parity Check) code, which are already proposed for NAND flash-based memories[26] can be used in NVM based systems as well. Moreover, modern server memory systems employ Chipkill-Correct as a standard feature to tolerate multi-bit errors as well as entire chip failure by distributing the codewords in multiple chips. Additionally, NVM-based main memories can be expected to have other reliability features (e.g., software-based redundancy) for higher reliability. Permanent errors (generally due to wear out of the cells) can also be prevented by either mapping the faulty cells using row sparing or column sparing techniques during production time[30], or even during field operation if Post Package Repair (PPR) is used. NVM memories are generally equipped with wear-leveling techniques[33] to further delay the wear out of the cells. Note that the choice of the ECC scheme is related but orthogonal to our work. We leave that decision for the system designer and assume that NVM modules would come with required ECC and other reliability features based on the RAS requirements.

2.4 Data Confidentiality in Improved Security NVM systems

Unlike DRAM, NVMs can retain data for a long time after power loss. As a result, it is possible to perform cold-boot attacks [19] in NVM without even freezing the memory. It is, therefore, always necessary to encrypt the contents stored in NVM. As discussed in the state-of-the-art approaches [4, 6, 45], counter-mode encryption is more suitable in protecting persistent Non-Volatile memories compared to other methods (i.e., Direct Encryption). In counter-mode encryption, each memory block is encrypted/decrypted using a unique One Time Pad (OTP) generated from a counter. Each encryption requires the creation of the OTP from an Initialization Vector (IV), which contains the counter, address of the memory block, and padding. While creating the OTP using the counter, the memory controller also brings the data block from the memory to be decrypted in parallel. Such concurrency of data fetching and OTP creation hides the decryption latency in counter-mode encryption. Figure 1 shows the encryption and decryption operations in counter-mode encryption.

Counters should be sized adequately to prevent overflows, which could cause the reuse of the same counter and hence OTP. State-of-the-art proposals employ different counter organizations in NVM to achieve different purposes. In ToC, each counter block contains eight counters and one MAC[16]. Such an organization allows counters to be treated as part of the Merkle-tree[49]. Split-counter scheme [44], on the other hand, packs 64 counters in one block by splitting counters into major and minor. Although this scheme allows more counters to be cached, every time a minor counter overflows, the major counter is incremented, and the memory controller needs to re-encrypt the whole page using the new major counter. Morphable counter proposes packing more counters in one block [36].

2.5 Integrity Protection

In a secure environment, fetching data from memory should be followed by an integrity check to make sure that the data has not been altered. In order to provide such integrity protection, the Merkle tree is used. Practical NVM sizes are in the order of Terabytes (TB) and require an enormous number of counters to protect each 64 Byte block in the memory. Protecting both data and counters in NVM may require a tree of an extreme size, which can be difficult to maintain. Prior work [44] shows that it is possible to reduce the tree size by protecting only the counters using a Merkle tree, while data values are protected by a Message Authentication Code (MAC) calculated over the data and the counter. Note that counters are stored off-chip, generally in NVM. While it is possible to relax their confidentiality, they need to be integrity-protected using the Merkle tree.

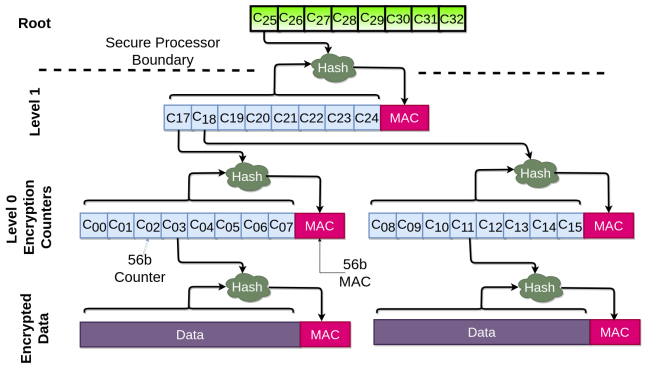

Figure 2: SGX's Tree of Counters (ToC) organization[16, 41, 49].

Figure 2 shows a simplified Tree-of-Counter currently adopted in industry [16]. In this type of tree, additional counters are introduced in the intermediate nodes of the tree. Such counters are incremented whenever the child node changes, and the MAC[1] value with each node depends on its own counters and upper-level counter. Such inter-level dependency using counters enables parallel updates where multiple MAC values can be calculated at once. Figure 2 shows a simplified version of such a tree used in Intel's SGX. Prior works [40, 41] have reduced the storage overhead of such tree by

[1] Authenticated Encryption engines such as AES-GCM[44] is typically used to ensure fast encryption, decryption, and MAC calculation. Since AES takes an address as an additional input, the blocks are protected from replay attacks.

integrating split-counter[44] style counter organization instead of using single monolithic counters. For instance, VAULT[41] uses 64-ary split-counters in the encryption counter level and variable-ary counters in the upper level. Similarly, we use 64-ary split-counters for encryption counters and plain 8-ary monolithic counters for the upper levels in our work. There is also another simplified Merkle-tree structure where the intermediate nodes are just hash values of the children [6]. While it is easy to recalculate such a tree merely from a child, they do not allow parallel tree updates. However, in the case of the parallel ToC, the tree can not be recalculated only from the children. A single uncorrectable error in the intermediate node can be of significant impact, leaving the system in either a vulnerable state if the error is tolerated, or losing valuable data covered by the node if everything is reinitialized.

Merkle trees can be either eagerly updated or lazily updated [49]. In the eager update scheme, the root of the Merkle tree is updated with each write, which means whenever an encryption counter is updated, the whole branch of the counter should be updated to reflect the new change. Although this updating scheme guarantees the freshness of the MT root and allows a single point of verification for memory integrity during the recovery [49], it incurs an extreme slowdown of the system. On the other hand, the lazy update scheme only updates the parent node on eviction, and updates lazily propagate upwardly. In other words, if an encryption counter is updated, nothing else is updated until the dirty counter gets evicted. Once a dirty counter is evicted, the parent node is fetched (if already not in the cache) and updated in the cache. This updating scheme reduces the performance overhead and the extra writes. However, a lazy update can make the root stale; thus, it requires an additional recovery scheme to recover from the system crash [49].

2.6 Crash Consistency

Secure memory operation with counter-mode encryption and Merkle tree requires access to up-to-date and verifiable encryption counter and tree node at any point of execution, either from the cache or from the main memory. A system crash may cause inconsistency between data, encryption counter, tree node, and the root if updated metadata is lost from the volatile cache before persisting their updates to memory. Strictly persisting metadata directly to the main memory guarantees that the memory copy is up-to-date and coherent with the tree root, however, at the cost of extreme performance overhead. Nevertheless, an efficient, high-performance and improved security NVM systems must use a fast volatile metadata cache. Osiris [45] solves this recoverability problem by encoding the encryption counter information with the ECC that can be later used as a sanity check of the counter. By limiting the maximum number of updates for a counter inside the cache, Osiris performs a limited number of counter trials during recovery and recovers the lost updates to the counter. This process must be followed by a Merkle tree regeneration, which verifies the validity of the correction process by matching the stored tree root with the newly calculated root.

While Osiris is limited to the BMT-style Merkle tree and has a time-consuming recovery process (needs to check every encryption and re-calculates all MAC values), Anubis[49] allows recovery of both types of trees within seconds. Anubis achieves such fast recovery with additional mechanisms that track the use of metadata cache and stores the tracking metadata in a shadow cache inside persistent memory. For each significant state change in the volatile cache, Anubis persists the address of the updated metadata so that later the recovery can be attempted only in the affected regions, not the whole tree. During recovery, Anubis first verifies the integrity of the shadow cache using a shadow cache and corrects all lost metadata. Note that none of the schemes allow for recovery if the metadata is found to be corrupted and the ECC fails to correct it.

2.7 Motivation

Improved security NVM systems are less reliable because errors in security metadata can jeopardize secure and verified operation on a large portion of the memory. It is difficult to simply ignore affected metadata or poison such metadata for their importance. In particular, in the case of ToC, where errors cannot be tolerated, such errors can leave the system in a vulnerable state. Depending on the location of errors and the metadata class, the impact of errors can be of various degrees. For example, losing an intermediate node due to error can leave a large portion of memory unverifiable depending on its position in the tree hierarchy. To better understand the problem, we study the expected[2] amount of lost/unverifiable data due to the different number of uncorrectable errors. To keep our discussion technology-independent, regardless of the source of uncorrectable errors, whether it is due to resistance drift (as in MLC PCM[3, 24, 48]), read/write disturbance errors, or other causes of incorrectness in metadata, we just vary the number of unexpected errors and compare the affected amount of data in secure versus non-secure 4TB memory system, as shown in Figure 3.

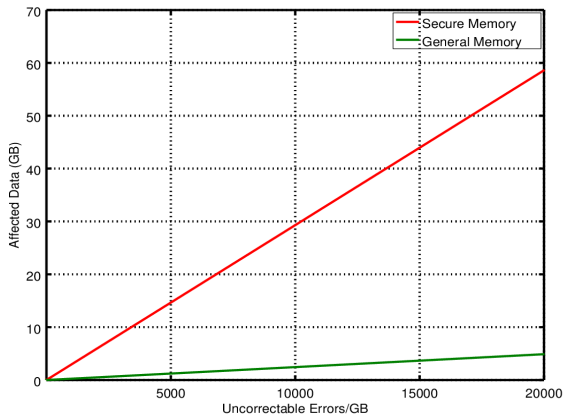

Figure 3: Impact of errors in metadata for secure memories.

In Figure 3, the expected amount of lost (or unverifiable) data in secure memory is 12x larger than that of general (non-secure) memory. Clearly, this makes secure systems 12x less resilient to faults compared to conventional systems. For instance, if our

[2]Expected amount is calculated through summing up the probability of error on each level of Merkle Tree multiplied by the amount of data rendered unverifiable, in other words, $E[X] = \sum_i X_i \times P(X_i)$, where X_i is the number of blocks lost due to error in level i and $P(X_i)$ is the probability of error in level i.

Table 1: Choice of tree update, persistence, and recovery scheme for Soteria.

Tree Update	Persistence Scheme	Recovery Mechanism
Lazy Update (ToC)	Update metadata on eviction, update shadow cache if metadata are modified in cache.	Restore metadata cache with the help of shadow cache. Verify shadow cache's integrity using shadow tree.

system has been subjected to high error/faults, long-duration power loss that caused resistance drift (if MLC PCM [3, 24, 48]), or any other causes of errors, we expect the secure system to lose 12x more data in comparison with the non-secure conventional system. Note that the inflation of the amount of lost data due to errors in the integrity tree is directly proportional to the number of levels; while level i in n-ary tree has n^i less nodes than leaves (assume data blocks), and hence n^i smaller probability of being affected by the error, it has an impact on n^i data blocks. The expected amount of data lost in a level is the same for all levels. In other words, even though the number of nodes in the Merkle tree is much smaller than regular data, each level in the Merkle tree adds the same possibility of losing a data block as the same possibility of losing data due to errors in regular data blocks. Therefore, the expected amount of lost data due to uncorrectable errors in integrity-protected memory is roughly nx that of the non-secure memory system, where n is the number of levels. With NVMs expected to be deployed in terabytes, or even petabytes for large-scale memory-centric systems [23], the tree depth can be in the range of tens of levels.

Obviously, the number of levels in the Merkle tree, which is a function of memory size (expected to grow), proportionally decreases the overall system resilience to memory faults. However, we observe that the number of nodes in the upper levels of the tree is much smaller than leaves, and their importance (coverage) is inversely proportional to both the frequency of their reads/updates and their storage overheads (number of nodes). Accordingly, Soteria aims to provide novel architectural support to improve the fault-tolerance of integrity-protected systems leveraging the architectural behavior and layout of Merkle trees.

3 SOTERIA DESIGN

In this section, we present the design details, the insights behind Soteria's design and challenges. Soteria inherits from the existing state-of-the-art implementations in improved security NVM systems, which results in improved security, crash consistency, and performance-friendly operation. The purpose of Soteria is to provide simple and performance-friendly augmentation on top of the existing security primitives to enhance metadata reliability.

Table 1 shows the tree update scheme, persistence scheme, and recovery mechanisms used by Soteria. Note that for counter recovery, we use the state-of-the-art scheme, Osiris [45]; however, any other schemes to recover counters (e.g., phasing[45]), can be used.

3.1 Decoupling Metadata Reliability from Data Reliability

Data in ECC DIMMs are protected from errors using data reliability schemes that can withstand a failure of a limited number of bits or chips. This error correction capability also covers the security metadata, which is also stored in the main memory. The strength of the existing data reliability schemes depends on the available resources (e.g., redundant storage, additional chips) and acceptable performance penalty. The ECC's strength is usually determined by considering the target RAS and acceptable hardware cost to achieve it. Generally, errors in the user application's data in memory do not cause a failure of the entire system; they may cause a termination of the application or loss of users' data. If additional reliability is required for some critical system software data (e.g., kernel code), additional precautions and measures can be taken without extending ECC's strength naively. For instance, modern servers allow Address Range Mirroring (ARM) [13] to duplicate kernel codes in the mirrored region. Such a decoupled reliability scheme allows for higher protection of critical data and general protection for all memory data.

Similarly, Soteria decouples the metadata reliability from the existing ECC. As such, assuming the existing ECC uses all additional bits per cache line, Soteria must store the additional reliability metadata for the security metadata elsewhere. The reliability metadata for security metadata can be either duplicate of the metadata (redundant copy), or long ECC codes stored in a separate memory region (Figure 5). Either way, an additional write per metadata write is required to update the reliability metadata for security metadata. For additional stronger ECC, additional code generation is required before writing the metadata. If performance is compared, both schemes will incur a similar number of writes and the same performance overheads (ECC calculation can be done while the original data is being written). However, the second option has hardware overheads for the ECC circuitry. Additionally, the stronger ECC for metadata needs to be stored in the data region, protected by a weaker version of ECC, which is counter-intuitive. The additional complexity and hardware cost for stronger ECC can be justified if and only if it protects the entire memory. Moreover, NVM capacities are expected to be large (in TBs), and the storage overhead for counters and Merkle-tree is already small enough, making the duplication a more cost-effective and straightforward solution. We use a split-counter scheme for the encryption counters, which packs 64 small minor counters, one major counter in each counter block similar to [9]. The storage overhead for counters is thus $\frac{1}{64}$, or (1.56%). The tree node in the above level is 8-ary and consumes even less storage. For instance, the storage overhead of the parent level of the encryption counter is only $\frac{1}{512}$, or (0.19%). The overhead for all the upper-level nodes is only (0.02%). In total, approximately 1.78% storage is required to accommodate the entire ToC in Soteria. Therefore, direct duplication of the metadata can provide a simple and effective solution to the problem.

3.2 Resilient-Security in Soteria

Our work's key observations are that the reliability and resilience of a secure system with encryption and integrity protection can be improved without augmenting or adding existing in-memory reliability features. Minor changes in the already existing security implementation inside the on-chip memory controller can negate the probability of errors in the metadata and significantly improve the resilience. We notice that while the importance of a metadata node increases as its position moves upwardly in the tree, the cost

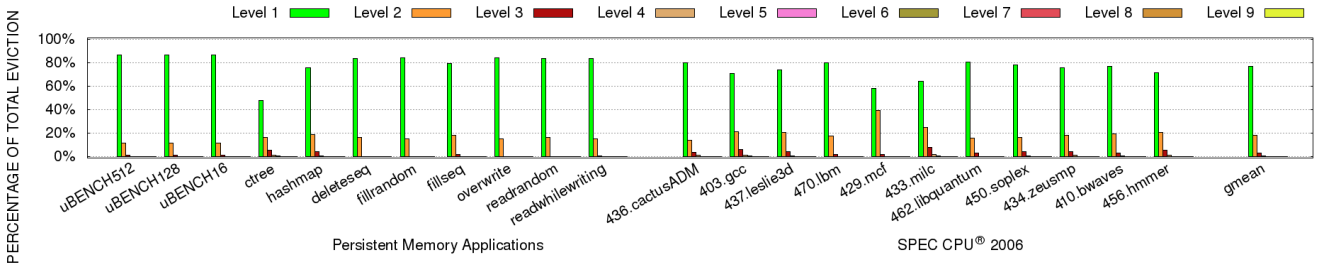

Figure 4: Percentage of eviction from different Merkle-tree level in lazy update.

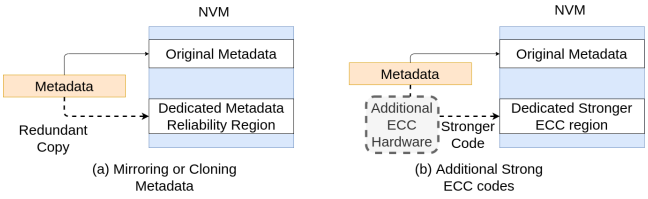

Figure 5: Redundant Metadata vs Strong ECC.

to make redundant copy reduces. The upper-level nodes have a very marginal storage overhead. For instance, the storage overhead of top four levels in an 8-ary tree protecting 1TB NVM ranges from $2.3e^{-5}$% to $4.6e^{-8}$%. In a lazily updated ToC tree, the frequency of the higher-level nodes' updates is also minimal. In conventional secure memory architecture, where a write-back metadata cache is used, metadata nodes are only updated when they are evicted from the cache. Figure 4 shows the frequency of evictions in different levels of the Merkle tree. It is clear from the figure that in a lazily updated tree, the upper-level nodes are rarely updated, and hence it is possible even to store multiple clones that can improve the availability of metadata significantly without extreme storage overheads (upper levels have a small memory footprint) and negligible performance implications (upper-level nodes are modified rarely).

Table 2: Soteria Metadata Cloning Depth for SRC and SAC.

	L1 (leaf)	L2	L3	L4	L5	L6	L7	L8	L9
SRC	2	2	2	2	2	2	2	2	2
SAC	2	2	3	3	4	4	4	4	5

3.2.1 Soteria Metadata Cloning (SMC). Figure 6 shows the organization of a simplified Merkle tree with *Soteria Metadata Cloning (SMC)*. SMC's key design goal is to have duplicate metadata (encryption counters and tree-nodes) always available. For such, metadata blocks will have clones stored in a separate region in NVM. In addition to the original node, every node can have *Primary Clone* and multiple *Reserved Clones*. The total number of copies, including the original node, is termed as the *depth* of the node. With more clones, a node has better chances of survival. We choose two modes of cloning, a 'Relaxed'cloning where all nodes have only a single additional clone regardless of their position in the tree, and an 'Aggressive'cloning where upper-level nodes have a higher number of clones. Table 2 lists two *Soteria Metadata Cloning* flavors with various depth in different levels in a 9 level (excluding the root) Merkle-tree, which can cover up to 1TB memory. *Soteria Relaxed Cloning (SRC)* is the least aggressive version, and *Soteria Aggressive Cloning (SAC)* is more aggressive in terms of maintaining high resilience. The most critical nodes that are stored in memory are the root's eight immediate children, each covering 12.5% of the memory. While determining maximum redundancy for the most critical nodes, we take into account the minimum WPQ (Write Pending Queue) size, which can be as low as 512B (eight entries)[42]. While an extremely high number of cloning is possible as top nodes are evicted rarely, the commit of clones must be broken down into multiple chunks if WPQ size is less than the number of generated clones, and may fail to commit all clones if a system crash occurs. Furthermore, a secure and recoverable NVM write may generate a maximum of three writes (cipher, data MAC and Shadow log) per write. Note that if the number of free entries in the WPQ is less than the number of generated clones because of some residue entries from previous writes, the memory controller will eventually be able to atomically commit all clones as soon as few entries are flushed from WPQ to NVM. On the other hand, if the number of clones is greater than WPQ size, it will always be difficult to commit all clones atomically. Although it is possible for a system to have a larger WPQ, we consider minimum WPQ size as well as some residue entries from previous writes, and conservatively set the maximum cloning to five. The number of clones for the rest of the intermediate nodes determined for SAC is below five and based on empirical analysis of the metadata cache eviction rates (Figure 4) as well as the coverage of the node. The two lowest levels contribute to more than 10% of the evictions while each node cover less than only 100kB data. Hence, no additional clones are made for them beyond what SRC already does. The next two levels contribute to only 1% to 10% of the total evictions, and hence SAC adds one additional clone. The rest of the levels contribute to less than 1% of total evictions and have two additional clones in SAC (Table 2). A system can be programmed to allow a different level of SACs and can be activated based on the desired reliability. Here, we only analyze a single SAC example based on our empirical analysis and compare it with SRC.

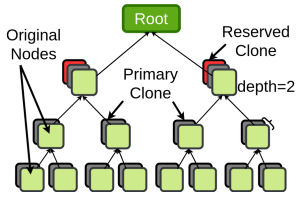

Figure 6: Soteria Metadata Cloning (SMC) Tree Structure.

Duplication in Soteria can be integrated easily with the ToC (Tree of Counters) style parallel tree as it is possible to use the Lazy update scheme with such tree structure to minimize the number of extra writes in the upper levels. Having duplicated nodes in the ToC tree is particularly important as otherwise, recovery would fail in the presence of error (unlike BMT, where intermediate nodes can be calculated from the children). Figure 7 shows the overall operation involving SMC. Data blocks evicted from LLC are encrypted, and respective counters and MT nodes are fetched from NVM or metadata cache for encryption and integrity verification. During this process, some metadata blocks are evicted from the cache. In such an event, apart from placing the evicted metadata block in the WPQ, it is also given to SRC for cloning. Eventually, all persistent writes (metadata eviction, SMC writes, shadow writes, and other persistent writes) will be placed in WQP. The Asynchronous Dram Refresh (ADR)[14] guarantees that contents in WPQ are persisted even if power fails. Note that WPQ size is limited to only tens of entries (e.g., 8 to 64) and is used to guarantee the atomicity of writes.

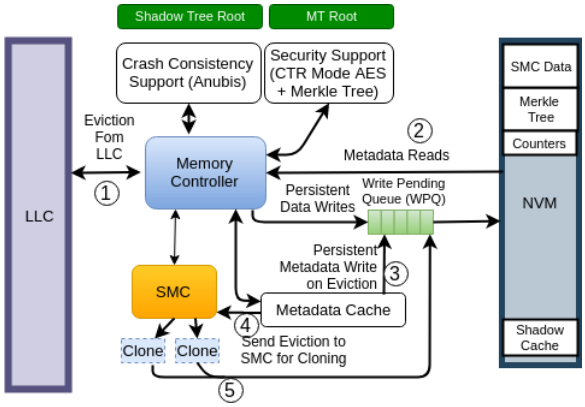

Figure 7: Soteria Metadata Cloning (SMC) design.

Cloning Shadow Cache: It is essential to track the metadata cache in ToC to provide both crash recoverability and fast recovery[49]. Soteria allows metadata to be updated inside the cache and only duplicates when they are evicted. Since Soteria adopts security, persistence, and crash consistency implementations from the state-of-the-art schemes, it is necessary to protect additional metadata (e.g., Shadow Cache) from unexpected uncorrectable errors for such schemes to preserve their capability.

Figure 8 shows the original shadow entry design and how we recondition the shadow structure entries within each block to have duplicate addresses and counter LSB clones. In the original structure, each entry in the shadow-cache consists of the address of the updated metadata (to identify the lost node), LSBs of the tree node counters (to fix the node counters), and a MAC value calculated over the node (to prevent replay within the shadow entry). Although such a design can effectively restore the system securely, the shadow-cache region's error can fail the recovery. The modified shadow-cache allows having duplicate entries without using extra space to achieve higher reliability. Note that ECC code-words are separated in eight chunks for correcting data read and write in word. We duplicate the information in each entry and make sure

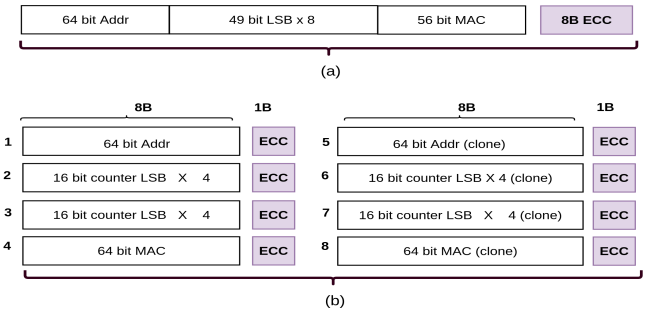

Figure 8: Modified shadow structure for duplicated entry. (a) Original Shadow Entry in Anubis, (b) Duplicated Shadow Entry.

that the duplicates are not within the same codeword protected by the same ECC check bits. Additionally, Soteria reduces the size of the LSB stored in Shadow Cache from 49 bit to 16 bit. Reduction in LSB size does not affect the performance since a consecutive update of a tree node's counter in the cache for 2^{16} times without being evicted from cache is extremely rare. Nevertheless, the node can be simply written back to memory to update the stale memory version in such a case. During recovery, a mismatch in MAC can identify the error, and it can be a straightforward process to fix the incorrect part using the correct one.

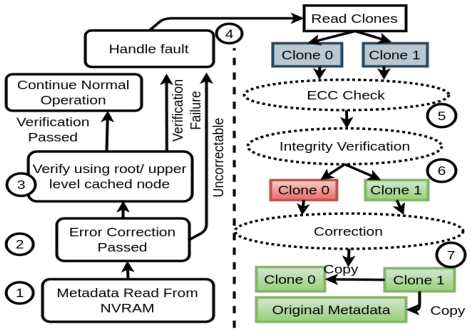

Figure 9: Fault Handling in Soteria.

3.2.2 Metadata Fault Handling Mechanism. Figure 9 shows Soteria's metadata fault handling method. A fault in a specific Merkle-tree node might happen for several reasons. Therefore, it is always expected and usual to check the contents for errors whenever they are read from memory. Soteria adds an additional level of correction by leveraging the data stored in the duplicate nodes. Whenever a memory version of a node is read, it is verified using the parent node (step 3). A mismatch in the MAC indicates that the reason could be a possible error/fault (if not a replay attack) and needs correction. If ECC alone cannot correct the node and an uncorrectable error occurs, Soteria brings all of the clones from memory and attempts to verify/repair it (step 4). Since it is improbable for all of the clones to be uncorrectable, one of the nodes would pass the verification check (step 6) and can purify all of the affected nodes.

While Soteria's primary goal is improved error tolerance of the security metadata, it must preserve the ToC's resistance to replay attacks. In the baseline ToC with no clones, the replay is detected

Table 3: Configuration of the simulated system.

Processor	
CPU	4 Cores, x86-64, Out-of-Order, 2.67 GHz
L1 Cache	Private, 2 Cycles, 32kB, 2-Way
L2 Cache	Private, 20 Cycles, 512kB, 8-Way
LLC	Shared, 32 Cycles, 8MB, 64-Way
Cacheline Size	64B
DDR Based PCM Main Memory	
Simulated Capacity	16 GB
PCM Latencies	Read 150ns, Write 300ns
Encryption Parameters	
Encryption	AES Counter Mode, 64-Way Split Counter[41]
Merkle-Tree	ToC style, Arity=8
Metadata Cache	512kB, 8-Way

by recalculating the MAC and comparing it with the existing value. To replay an old MAC, the recalculation must be done using the older version of the counter, which is protected by a separate MAC. In other words, the attacker will have to replay old data-MAC, counters, and counter-MACs in all above levels and the root of the Merkle-tree to replay old data. A replay of the intermediate node is also detected in the same way. While Soteria makes no change in the tree design, it only introduces redundancy for nodes. If an attacker replays a pair of old data and MAC, it will be similarly detected in Soteria. However, since there are multiple duplicates of the intermediate nodes, replaying a single MT node will end up being corrected by Soteria. If the attacker replays all clones of a node, Soteria's recovery will fail in the integrity verification stage (step 6), and the attack will be detected unless the attacker is able to identify collisions. Soteria keeps the MAC size (64 bit) unchanged and hence the collision rate is also similar to the prior works [5, 6, 37, 49].

4 EXPERIMENTAL METHODOLOGY

We evaluate Soteria from both performance and reliability point of view. For performance evaluation, we use Gem5[7], a cycle-accurate simulator. We modify the memory controller and memory access latency to model NVM based memory system. We add security implementations inside the memory controller (encryption, integrity protection, Anubis tracking, and Soteria). The Gem5 simulation parameters are listed in Table 3. We use a number of persistent and non-persistent memory applications to evaluate our scheme. In addition to several standard persistent memory benchmarks (Whisper[31] and PMEMKV[1]), we also use some in-house micro-benchmarks that sequentially access a large array stored in NVM. `uBENCH X` accesses one byte after every `X` bytes in sequential manner with read/write ratio of 1. Additionally, we use SPEC CPU 2006 benchmark to represent typical non-persistent memory applications[3]. Note that the security metadata must be updated for both persistent and non-persistent applications to ensure security and consistency. Since we assume transparent memory protection in the memory controller, Soteria treats most applications in substantially similar manner and the performance depends on the application's memory access pattern. We create checkpoint each application after initialization phase and simulate 500M instructions afterwards.

For evaluating the resilience to errors of our proposed scheme, we use an industry-grade memory reliability simulator, FaultSim[34].

[3]SPEC®, SPEC CPU®, SPECint®, and SPECfp® are registered trademarks of the Standard Performance Evaluation Corporation. SPEC CPU 2006 has been retired, and SPEC® is no longer reviewing or publishing results with that benchmark. More information about SPEC CPU 2006 can be obtained from the SPEC website [2].

Table 4: Faultsim Configuration.

Chips, Chips/Rank, Bus Per Chip	18,9,8
Rank, Bank, Rows, Cols	2,16,16384,4096
Repair Mechanism	Chipkill
Failure Distribution	Hopper [39]
FIT	Varied to Get Sensitive Analysis
Data Block	512 bits
No of Simulation	1 Million

NVM-based main memories are still in their development phase, and large-scale field study of their failure rates is still not available. However, we keep in mind that NVMs are likely to be designed to meet the RAS requirements similar to current DRAM systems. To overcome this shortcoming, and to keep our evaluation focused mostly on analyzing the effect of errors in security metadata, we use modern DRAM's failure rate in our simulations. We use failure distribution for single bit, single word, single column, single row, single bank, nBank, and nRank reported in [39] for Hopper supercomputer. However, we keep in mind that NVMs are expected to be less reliable compared to DRAM and perform sensitivity analysis by varying the FIT values from low to high to model a variety of reliability scenarios due to differing NVM technologies.

The FIT values captured in our simulation are chosen by keeping the modern large-scale server's failure rate. Saurabh et al. show in their study [17] that MTBF (Mean Time Between Failure) of several large-scale servers ($\approx$ 20k nodes) range from 7 hours to 23 hours. We, therefore, calculate MTBF for each FIT/Deice values obtained in our evaluation by calculating the total failure rate of the system with 20k nodes, 4 DIMMs per node, and 18 chips per DIMM. We can then verify the MTBF of the simulated system against the reported MTBFs in the above-mentioned paper to keep the assumptions about failure rates within a practical limit. Our calculated MTBF ranges between 694 Hours (1 FIT) to 8.6 Hours (80 FIT). We vary the failure rate from low to high during our evaluation and only keep the data point that satisfies the MTBF reported in prior studies. Table 4 presents the FaultSim simulator configuration in our experiments.

5 EVALUATION AND RESULTS

In this section, we analyze the performance and resilience of Soteria. Here we first study the performance impact of adding Soteria in a secure system; later, we discuss the resilience analysis.

5.1 Metadata Cache Eviction Characteristics

For evaluating Soteria, we have simulated a secure x86 processor with lazy-update ToC-style Merkle-tree in Gem5 along with necessary persistency and crash consistency measures. Since Soteria activates cloning only during an eviction, it is important to understand the application's eviction behavior in the lazily updated ToC tree. Figure 10c shows eviction characteristics of the workloads used in our evaluation. It is clear from the figures that the rate of the evictions is very low (approximately 1.3% of total memory operations), and most of the evictions are captured in the lower level in a lazy-style tree update (Figure 4). Upper-level nodes get updated/modified rarely, and hence their eviction rate is also significantly low compared to the lower-level nodes. Workloads that are less memory intensive show evictions mostly from the bottom level, where memory-intensive applications demonstrate comparatively

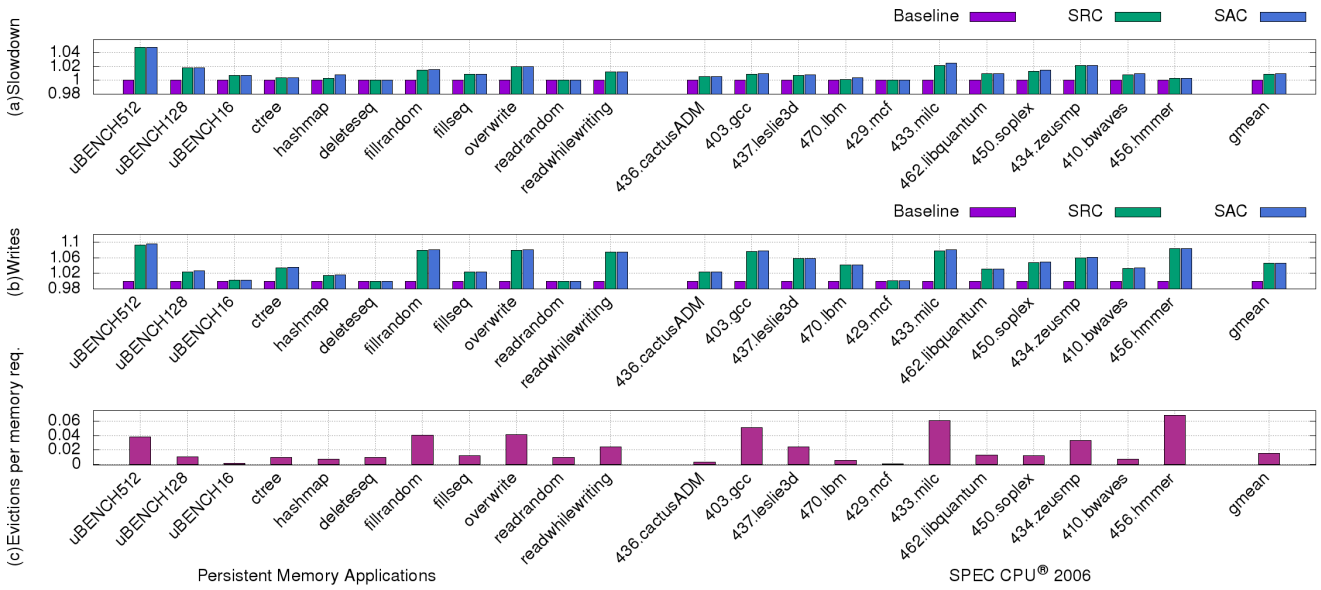

Figure 10: (a) Performance overheads of Soteria schemes. (b) Writes behavior. (c) Metadata cache eviction per memory request.

increased eviction from the intermediate level nodes. However, we observe that even for memory-intensive applications, it is rare for the highest level (top 3/4 levels) nodes in the tree to be updated or get evicted from the cache as updates only propagate lazily towards the root.

5.2 Soteria Metadata Cloning

Figure 10 shows the overheads for Soteria Metadata Cloning (SMC). The following are the details of the schemes compared in Figure 10.

- **Baseline:** The baseline here is a improved security NVM systems (according to state-of-the-arts [49]) memory architecture with ToC-style Merkle-tree with the lazy update. Cache tracking mechanism similar to Anubis [49] is also implemented within the baseline to account for the recovery.
- **Soteria Relaxed Cloning (SRC):** A duplication mechanism with a lazy tree update scheme, where every node is duplicated only once (Table 2) whenever evicted from the cache.
- **Soteria Aggressive Cloning (SAC):** A duplication mechanism where upper-level nodes are duplicated more than once (Table 2).

From Figure 10a, it is clear that Soteria does not incur expensive overhead. Aggressively cloning upper-level Merkle-tree nodes barely adds extra overheads. This is due to two fundamental facts. First, the cache miss rate for metadata cache for Merkle-tree node is minimal (less than 4% for most applications). Second, the use of the lazy scheme reduces the extra writes in the upper-level Merkle-tree nodes. Since the metadata cloning is done during evictions, the percentage of extra writes and performance slowdown is related to the application's eviction behavior (Figure 10c). For instance, uBENCH16 and uBENCH128 both access an array sequentially, but the latter incurs more evictions due to the larger stride and higher cache miss. In applications that are highly read-intensive (e.g., 429.mcf), the metadata evictions are a minute portion of total memory operations (all reads and writes), and hence Soteria does not add any significant overhead over the baseline. Additionally, SAC only makes additional clones for nodes that are positioned higher in the tree hierarchy, which is significantly rare in the lazy eviction method. On average, Soteria Relaxed Cloning (SRC) incurs about 1% extra execution time overhead, and Soteria Aggressive Cloning (SAC) incurs about 1.1% execution time overhead on top of the baseline. The writes overhead in SRC and SAC are 4.3% and 4.4% approximately, as shown in Figure 10b.

5.3 Resilience Analysis

The system's reliability can be measured in terms of various metrics that can give different insights. For instance, MTTF (Mean Time To Failure) represents how soon a fault will occur, and MTBF provides insights on how frequently a fault will occur. Since every fault is treated equally in MTTF or MTBF, these metrics do not provide insights on the impact of a single fault (or uncorrectable error) on the system (e.g., amount of data loss). Soteria's goal is to minimize the impact of security metadata error on data for a given failure rate. For quantifying the resilience to security related data loss, Soteria considers existing metrics (e.g., FIT), and calculates the percentage of data loss for a given failure rate.

To evaluate the reliability in terms of the amount of data loss, we measure how much reduction in post-error unverifiable data can be achieved with Soteria. Data loss in our evaluation can be classified into two general classes, L_{error}, which is the data loss due to memory error, and $L_{\text{unverifiable}}$ which happens when we cannot verify data (which can be error-free) because of error in the respective security metadata. Metadata errors can show up within days (in case of weak ECC) or months (in case of stronger ECC) if the entire memory (expected to be in Terabytes for emerging NVM) needs to be protected, requiring a large metadata region. With Soteria, we can expect a significant delay in such an event since the probability of having

all clones of the metadata as uncorrectable is a rare event. However, resilience to errors for the improved security NVM systems can be obtained from the percentage of unverifiable data (due to metadata error) in a system. The aim of Soteria is to reduce such unverifiable data percentage as much as possible. As such, in our resilience analysis, we define Unverifiable Data Ratio (UDR), which quantifies the survivability of the improved security NVM system in the presence of errors that can affect security metadata. UDR is simply the ratio of the unverifiable memory region to the total memory size.

$$UDR = \frac{L_{\text{unverifiable}}}{Total\ Memory\ Size}$$

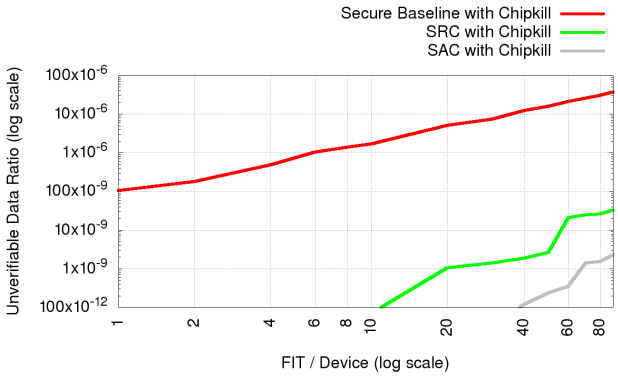

Figure 11: UDR comparison among secure systems with and without Soteria.

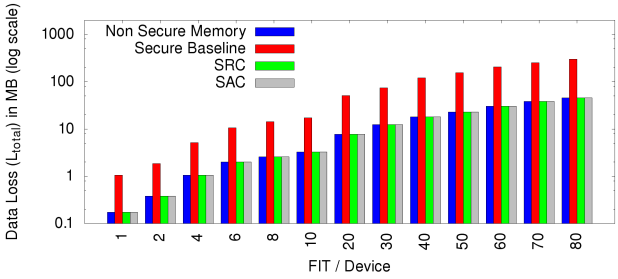

Figure 12: Unverifiable Data for an 8TB NVM Main Memory.

We use the FaultSim simulator to collect uncorrectable errors in the security metadata region for five years of simulated time in the presence of Chipkill as the underlying error correction scheme. We then calculate the total unverified memory region, considering the location of the affected metadata and their importance in the tree hierarchy. Figure 11 shows how UDR increases with a higher failure rate. In the case of the secure baseline scheme with Chipkill, small errors in the metadata cause a large portion of data to be unverifiable. As a result, UDR increases significantly with the increase of failure rate in a secure system without Soteria. However, since Soteria maintains one additional clone in SRC and aggressively clones the higher-level nodes in SAC, it shows significant resilience to errors. Unverifiable data loss will only occur if and only if all clones of security metadata in Soteria become faulty at the same time. Even for the maximum failure rate tested (FIT 80) with Chipkill-Correct, *SRC*'s UDR remains as low as at 2.66×10^{-8}, and *SAC*'s UDR remains as low as at 1.5×10^{-9}. On the other hand, the baseline secure scheme's UDR is 3×10^{-5} at FIT=80. For lower failure rate (FIT = 1 to 10), we have observed no data loss due to verification failure in Soteria, while the baseline scheme's UDR is observed to be more than the UDR captured for SAC and SRC in the highest failure rate. The geometric mean for the UDR reduction is 2.5×10^3 times for SRC, and 3.7×10^4 times for SAC, compared to the Secure Baseline. In other words, we can say SRC is on average 2.5×10^3 times more resilient, and SAC is 3.7×10^4 times more resilient compared to the baseline.

Figure 12 shows how UDR values can be translated to a practical NVM size of 8TB. The total data loss L_{total} is the summation of L_{error} and $L_{unverifiable}$. The Non-Secure Memory loses data due to only memory errors ($L_{total} = L_{error}$). With the introduction of security metadata in the Secure Baseline scheme, it loses 5.06X more data for verification failures that occurred due to errors in the security metadata. SRC and SAC can significantly reduce the $L_{unverifiable}$ and keeps L_{total} extremely close to L_{error}. Note that, with the failure rate assumed, the difference in data loss for SRC and SAC is minute if compared against the large value of L_{total}. However, Figure 11 shows that SAC is on average 20X more resilient to errors compared to SRC, which can be used in large-scale systems where the accumulated memory size is extremely large.

6 DISCUSSION

In this section, we discuss the security and reliability aspects of our scheme.

6.1 Security Discussion

With Soteria primarily focusing on having a more reliable and secure system by minimizing the unverifiable data, it is also essential to analyze its impact on security. ToC-style Merkle-tree is generally considered more secure compared to BMT as the counters in the intermediate nodes are protected by two MAC values (one within the node itself and the other in the parent node protecting the parent counter). Therefore, it is rare to find collisions and difficult for the attacker to replay any node without modifying the whole tree. However, to achieve crash consistency, state-of-the-art works[49] lazily update the tree, which can make the root deviate from being up-to-date, therefore, makes the metadata cache its trust base and protect its integrity using an eagerly updated small BMT-style tree. This prevents replay attacks on the shadow region in Anubis and makes the system secure even if the root is not eagerly updated. Soteria adopts the same security implementation in ToC-tree while making duplicate entries of the shadow region in the memory, making it more resilient to errors. Soteria is as secure as its baseline, if not more, with such duplicate metadata region residing in a different memory location. We choose ToC over BMT because of the parallel update capability, improved security, and industry adoption of ToC. However, if BMT is used, similar concepts can be applied to the encryption counters.

6.2 Reliability Discussion

Reliability analysis in section 5 clearly shows that Soteria can reduce the data loss occurred by errors in the security metadata significantly. One important thing to note here is that we carefully

design our system to be completely decoupled from the existing reliability features. Soteria can work with any underlying error correction mechanism and can significantly amplify the survivability of security metadata. The data loss due to security unverifiability in improved security NVM systems due to the sensitivity of security metadata is a problem that must be addressed by re-designing security metadata management implementation, not strengthening the ECC of the system. An ECC implementation is designed to provide a reasonable level of reliability without considering the security-related issues. Although stronger ECC will be able to reduce the effect of errors in metadata, our analysis shows that Soteria with baseline ECC can provide better survivability of security metadata compared to a stronger ECC working alone. Another possible design option can be storing stronger code (e.g., multi-bit correction) for metadata instead of storing additional clones. This design option would incur similar write overhead and can also improve reliability for security metadata; however, it would require additional hardware for encoding and decoding the extra ECC code and hence will require additional power. Soteria shows that simple duplication can augment the reliability significantly for security metadata with minimal cost. Note that such duplication is common for reliability enhancements, for instance, in Address Range Mirroring [13] techniques that mirror sensitive system software data (e.g., kernel space) for better reliability. Besides, we show that it is easy to achieve a higher level of duplication, and hence stronger reliability, with minimal performance and write overhead due to the lazy update style of security metadata.

7 RELATED WORKS

Although NVM reliability is not explored enough compared to how DRAM reliability is explored by industry and academia, its research potential and popularity among the researchers are growing fast as the industry is considering NVM as a potential solution to the DRAM scalability problem. Most NVM reliability works focus primarily on two directions. First, some works explore the solution to the endurance problem of Emerging NVM devices. Start-gap wear-leveling [33] is one of the pioneering work related to wear-leveling techniques to increase the endurance of the NVM cells. Second, some works focus on improving the error correction capability of the systems for both hard errors (which are mostly due to low endurance) and soft, or transient errors. Pay-As-You-Go or PAYG [32] provides a dynamic solution for mitigating hard errors, while Fan et al. in Aegis [15] use partitioning mechanism in the data block to recover data from stuck-at-fault. Many prior works also focus on soft error correction using a wide variety of ECC mechanisms. In general, the error correction efforts can be subdivided into two different approaches, where, in one approach strong Error Correction Code (ECC) is used to provide reliability to soft errors [48], the other approach looks at Error Correction Pointers (ECP) [38] for that purpose.

Although previously mentioned works should be in consideration for any NVM reliability works and are related to our work, our direction is slightly orthogonal to these works since Soteria focuses mostly on the impact on the reliability of secure systems because of security metadata. In the NVM security domain, previous works have looked at the efficient implementation of security, where the focus is mostly on reducing performance overhead and extra writes and providing crash consistency. Most NVM security exploration in research advocates for Counter Mode Encryption [4, 45, 49] and Merkle Tree [4, 45, 46, 49] for a secure memory system. Silent Shredder [4] aims to reduce the extra NVM writes in secure memory by an optimized shredding mechanism. Osiris [45], Anubis [49], Crash Consistency [35], Triad NVM [5] all explore the resilience of an NVM system to system crash and efficient recovery.

8 CONCLUSION

Soteria explores the reliability of Non-Volatile main memories operated in a secure environment. While errors or fault in NVM cells that store data have a small footprint of damage if compared to the entire memory, errors or fault in security metadata can leave a large portion of memory unreliable and unverifiable. Soteria improves the overall reliability of a improved security NVM systems without extreme error correction implementation and a severe performance penalty. We have shown that it is possible to deny the reliability threat posed by secure implementation very efficiently and without extreme performance overhead. Soteria improved security and reliability with approximately 1% performance overhead. In summary, Soteria substantially improves the resilience of the improved security NVM systems with minimal overheads.

ACKNOWLEDGMENTS

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA) and the Office of Naval Research (ONR). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. Approved for public release. Distribution is unlimited.

REFERENCES

[1] [n. d.]. pmemkv. "https://github.com/pmem/pmemkv". [Online; accessed 05-February-2020].
[2] [n. d.]. SPEC CPU 2006 Benchmarks. https://www.spec.org/cpu2006/. Accessed: 2019-11-13.
[3] Aravinthan Athmanathan, Milos Stanisavljevic, Nikolaos Papandreou, Haralampos Pozidis, and Evangelos Eleftheriou. 2016. Multilevel-cell phase-change memory: A viable technology. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 6, 1 (2016), 87–100.
[4] Amro Awad, Pratyusa Manadhata, Stuart Haber, Yan Solihin, and William Horne. 2016. Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers. In *ACM SIGARCH Computer Architecture News*, Vol. 44. ACM, 263–276.
[5] Amro Awad, Laurent Njilla, and Mao Ye. 2018. Triad-NVM: Persistent-Security for Integrity-Protected and Encrypted Non-Volatile Memories (NVMs). *arXiv preprint arXiv:1810.09438* (2018).
[6] M. Prvulovic B. Rogers, S. Chhabra and Y. Solihin. 2007. Using Address Independent Seed Encryption and Bonsai Merkle Trees to Make Secure Processors OS- and Performance-Friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*.
[7] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH Computer Architecture News* 39, 2 (2011), 1–7.
[8] Youmin Chen, Jiwu Shu, Jiaxin Ou, and Youyou Lu. 2018. HiNFS: A persistent memory file system with both buffering and direct-access. *ACM Transactions on Storage (TOS)* 14, 1 (2018), 4.
[9] M. Prvulovic B. Rogers Chenyu Yan, D. Englender and Yan Solihin. 2006. Improving Cost, Performance, and Security of Memory Encryption and Authentication. In *33rd International Symposium on Computer Architecture, ISCA'06.*

[10] Siddhartha Chhabra and Yan Solihin. 2011. i-NVMM: a secure non-volatile main memory system with incremental encryption. In *2011 38th Annual international symposium on computer architecture (ISCA)*. IEEE, 177–188.

[11] Chiachen Chou, Prashant Nair, and Moinuddin K Qureshi. 2015. Reducing refresh power in mobile devices with morphable ECC. In *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 355–366.

[12] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[13] Neo Cui. 2017. Demonstrating the Memory RAS Features of Lenovo ThinkSystem Servers. *Lenovo Press, Oct* 31 (2017), 25.

[14] Samantha J Edirisooriya, Shanker R Nagesh, Blaine R Monson, and Pankaj Kumar. 2017. Method and apparatus for completing pending write requests to volatile memory prior to transitioning to self-refresh mode. US Patent App. 14/816,445.

[15] Jie Fan, Song Jiang, Jiwu Shu, Youhui Zhang, and Weimin Zhen. 2013. Aegis: Partitioning data block for efficient recovery of stuck-at-faults in phase change memory. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 433–444.

[16] Shay Gueron. 2016. A Memory Encryption Engine Suitable for General Purpose Processors. Cryptology ePrint Archive, Report 2016/204. https://eprint.iacr.org/2016/204.

[17] Saurabh Gupta, Tirthak Patel, Christian Engelmann, and Devesh Tiwari. 2017. Failures in large scale systems: long-term measurement, analysis, and implications. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[18] Frank T Hady, Annie Foong, Bryan Veal, and Dan Williams. 2017. Platform storage performance with 3D XPoint technology. *Proc. IEEE* 105, 9 (2017), 1822–1833.

[19] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. 2009. Lest we remember: cold-boot attacks on encryption keys. *Commun. ACM* 52, 5 (2009), 91–98.

[20] Jim Handy. 2015. Understanding the Intel/Micron 3D XPoint memory. *Proc. SDC* (2015).

[21] John L Henning. 2006. SPEC CPU2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News* 34, 4 (2006), 1–17.

[22] HWPOISON. 2009. HWPOISON. "https://lwn.net/Articles/348886/". [Online; accessed 10-October-2019].

[23] Kimberly Keeton. 2015. The machine: An architecture for memory-centric computing. In *Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*.

[24] Win-San Khwa, Meng-Fan Chang, Jau-Yi Wu, Ming-Hsiu Lee, Tzu-Hsiang Su, Keng-Hao Yang, Tien-Fu Chen, Tien-Yen Wang, Hsiang-Pang Li, Matthew Brightsky, et al. 2016. A Resistance Drift Compensation Scheme to Reduce MLC PCM Raw BER by Over 100x for Storage Class Memory Applications. *IEEE Journal of Solid-State Circuits* 52, 1 (2016), 218–228.

[25] Linux. 2019. Linux Direct Access of Files (DAX). "https://www.kernel.org/doc/Documentation/filesystems/dax.txt". [Online; accessed 22-July-2019].

[26] Ren-Shuo Liu, Meng-Yen Chuang, Chia-Lin Yang, Cheng-Hsuan Li, Kin-Chu Ho, and Hsiang-Pang Li. 2014. EC-Cache: Exploiting error locality to optimize LDPC in NAND flash-based SSDs. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

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

[28] Sparsh Mittal. 2017. A survey of soft-error mitigation techniques for non-volatile memories. *Computers* 6, 1 (2017), 8.

[29] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. 2018. A comparison study of intel SGX and AMD memory encryption technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. ACM, 9.

[30] Prashant J Nair, Dae-Hyun Kim, and Moinuddin K Qureshi. 2013. ArchShield: Architectural framework for assisting DRAM scaling by tolerating high error rates. In *ACM SIGARCH Computer Architecture News*, Vol. 41. ACM, 72–83.

[31] Sanketh Nalli, Swapnil Haria, Mark D Hill, Michael M Swift, Haris Volos, and Kimberly Keeton. 2017. An analysis of persistent memory use with WHISPER. *ACM SIGPLAN Notices* 52, 4 (2017), 135–148.

[32] Moinuddin K Qureshi. 2011. Pay-As-You-Go: low-overhead hard-error correction for phase change memories. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 318–328.

[33] Moinuddin K Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. 2009. Enhancing lifetime and security of PCM-based main memory with start-gap wear leveling. In *Proceedings of the 42nd annual IEEE/ACM international symposium on microarchitecture*. ACM, 14–23.

[34] D Roberts and P Nair. 2014. FAULTSIM: A fast, configurable memory-resilience simulator. In *The Memory Forum: In conjunction with ISCA*, Vol. 41.

[35] J. Ren S. Liu, A. Kolli and S. Khan. 2018. Crash Consistency in Encrypted Non-volatile Main Memory Systems. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[36] Gururaj Saileshwar, Prashant J Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A Joao, and Moinuddin K Qureshi. 2018. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. In *Microarchitecture (MICRO), 2018 51st Annual IEEE/ACM International Symposium on*.

[37] Gururaj Saileshwar, Prashant J Nair, Prakash Ramrakhyani, Wendy Elsasser, and Moinuddin K Qureshi. 2018. Synergy: Rethinking secure-memory design for error-correcting memories. In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 454–465.

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

[39] Vilas Sridharan, Nathan DeBardeleben, Sean Blanchard, Kurt B Ferreira, Jon Stearley, John Shalf, and Sudhanva Gurumurthi. 2015. Memory errors in modern systems: The good, the bad, and the ugly. *ACM SIGPLAN Notices* 50, 4 (2015), 297–310.

[40] 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 *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 735–748.

[41] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. 2018. VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. 665–678.

[42] Zixuan Wang, Xiao Liu, Jian Yang, Theodore Michailidis, Steven Swanson, and Jishen Zhao. 2020. Characterizing and modeling non-volatile memory systems. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 496–508.

[43] Kai Wu, Frank Ober, Shari Hamlin, and Dong Li. 2017. Early evaluation of Intel Optane non-volatile memory with HPC I/O workloads. *arXiv preprint arXiv:1708.02199* (2017).

[44] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. 2006. Improving cost, performance, and security of memory encryption and authentication. In *ACM SIGARCH Computer Architecture News*, Vol. 34. IEEE Computer Society, 179–190.

[45] Mao Ye, Clayton Hughes, and Amro Awad. 2018. Osiris: A Low-Cost Mechanism to Enable Restoration of Secure Non-Volatile Memories. In *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2018)*.

[46] 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* (2019).

[47] Vinson Young, Prashant J Nair, and Moinuddin K Qureshi. 2015. DEUCE: Write-efficient encryption for non-volatile memories. *ACM SIGPLAN Notices* 50, 4 (2015), 33–44.

[48] Da Zhang, Vilas Sridharan, and Xun Jian. 2018. Exploring and Optimizing Chipkill-Correct for Persistent Memory Based on High-Density NVRAMs. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 710–723.

[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 International Symposium on Computer Architecture*. ACM, 157–168.