# RC-NVM: Recovery-Aware Reliability-Security Co-Design for Non-Volatile Memories

Kazi Abu Zubair, Rahaf Abdullah, David Mohaisen, and Amro Awad

**Abstract**—Non-Volatile Memory (NVM) technologies are now available in the form of byte-addressable and fast main memory. Despite their benefits, such memories require secure and reliable memory management to prevent malicious and spontaneous data alteration. However, in NVM security, it is still a major challenge to maintain crash consistency and reliable system recovery. In particular, Message Authentication Codes (MAC) are rarely discussed in recent recovery-aware NVM studies since they are generally not cached. MACs have outstanding sensitivity to memory errors and hence they can be used for reliability enhancement alongside their mainstream use to detect malicious tampering. However, persisting MACs is challenging and requires 2x writes and reads in a conventional secure NVM system. It is possible to cache MACs in a MAC-assisted reliability scheme; however, this brings many challenges related to crash consistency and reliability. In this paper, we present the difficulties associated with MAC recovery if they are cached, and solutions to guarantee reliable system recovery. Finally, we propose a novel scheme, **R**ecoverable and **C**hipkill capable **NVM**, RC-NVM, which can effectively use a volatile write-back cache for MACs as well as recover them quickly after a system crash. Our scheme reduces 27% of the writes and allows 18.2% performance improvement compared to the state-of-the-art, while preserving the ability to recover from a system crash.

**Index Terms**—NVM, Security, Crash Consistency, Reliability

## 1 Introduction

Byte addressable and fast Non-Volatile-Memory (NVM) technologies such as Intel's Optane DCPMM [22] are opening many possibilities in both general-purpose, as well as high-performance computing. However, although the non-volatility, byte-addressability, near DRAM speed, and high density of NVM make it a good candidate for future Universal Memory (UM [55]), it has many challenges. These challenges have motivated many ideas which try to solve many different aspects of NVM, such as wear-levelling [7], [33], [37], [40], [64], performance [23], [28], [65], reliability [26], [35], [46], [54], [63], [66], and security [5], [8], [9], [10], [20], [31], [57], [60], [61], [69], [70], [73], [74], all of which offer some unique advantages with minor changes in the processor, memory design, system software, and programming model.

A key challenge in Non-Volatile Main Memories is ensuring data security. For instance, due to the data remanence of NVM cells, previously stored data can be extracted from a stolen NVM device without even freezing memory to cryogenic temperature in contrast to DRAM [18], [47], [62]. The Operating System (OS) and applications may leave many information and traces in the persistent main memory[1]. This makes full memory encryption a primary requirement for secure Non-Volatile Memories. While encryption of the data ensures confidentiality, it must also come with data replay protection to guarantee that the integrity of the memory contents is preserved. Several prior works have proposed many different ideas to achieve **C**onfidentiality [4], [50], [56] and **I**ntegrity [10], [41], [44]—two fundamental aspects of the CIA (Confidentiality, Integrity, and Availability) triad. To meet such an ever-growing demand for a secure computing environment, the industry is also adopting confidentiality and integrity protection support to their processors [14], [17]. Until now, it is a well-established practice to use Counter Mode Encryption (CME) for confidentiality, and Merkle-tree with Message Authentication Codes (MAC) for protection against replay attacks.

Typically, the final aspect of the CIA triad, the **A**vailability, is ensured by using many different reliability features such as Error Correction Codes (ECC) [19], [35], [66], parity [13], [45], [52], [59], journaling and snapshotting [38], [53], and software redundancy [26], [54].

While many of these features can be added optionally in many different layers in the system software or the hardware, the fundamental and most common reliability feature in any high-availability memory system is the hardware-managed ECC. Such hardware-managed ECC performs two basic operations in the memory controller: error checking and error correction. Interestingly, as discovered in many prior studies [21], [45], [52], data MACs used in secure memories have an outstanding sensitivity to memory errors and hence can be used to detect them. This means that we can delegate the error detection to the MACs and use all available ECC bits for error correction. Such delegation of error detection responsibility to MAC provides stronger error detection and correction capabilities because all ECC bits can be employed for correction, and a strong cryptographic MAC is highly likely to detect data alteration due to errors. For instance, Synergy [45] proposes to achieve

• *K. A. Zubair, R. Abdullah, and A. Awad are with the Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC, 27606. D. Mohaisen is with the Department of Computer Science, University of Central Florida, Orlando, FL 32765.*

1. We consider NVM to be the main memory, which is visible to the application's address space, with or without an intermediate DRAM cache. Both persistent and non-persistent data can be stored in NVM.

chipkill-level reliability by using data MACs to detect errors and parities to correct an entire faulty chip. It observes that if MAC is co-located with the data instead of ECC in an ECC-DIMM, it is possible to check for both data errors and malicious modification using the same MAC without any extra memory read. If a discrepancy between calculated MAC and stored MAC is discovered, Synergy speculatively tries to correct each chip using the parity and verifies every time whether successful chipkill-correction was performed using the MAC.

While a system with such coordinated security and reliability can achieve higher resilience to errors, it has some practical limitations that must be addressed before its adoption. The first obvious limitation is the update of parities, which is done for every memory write. This incurs 2x writes and hence will significantly reduce the memory lifetime if used in NVM. The second limitation is the write-masking requirement for parity update. Since multiple parities are stored in a single cache line, updating a single parity requires read-modify-write of the parity cache line or write masking inside the DIMM. As observed by ITESP [52], both of these issues can be solved by using a dedicated cache that caches parity. With caching, there is no need for read-modify-write or write masking since the parity cache line in a write-back cache will be able to coalesce parity updates and commit all parity changes at once during its eviction. Clearly, using a dedicated cache to cache parity is an obvious solution–at least for DRAM-based systems.

**The Problem.** While both Synergy and ITESP are designed primarily for DRAM-based systems, such coordinated security-reliability can also be a useful feature in a secure NVM system. However, additional complexity arises during a system recovery after a crash if we use a write-back SRAM cache for the parities. Updated parities in the cache will be lost in a system crash, leading to a post-crash inconsistency between the data and the parity. Naively, such an event can be ignored, assuming that there will likely be no error for the crash-affected data lines, and we can simply re-calculate the parity after verifying the data. However, in reality, this will be an overly optimistic and rare case. Even if a single bit changes in a data cache line for which the updated parity is lost in a crash, there will be a MAC mismatch and no means for data recovery. The system will identify such an event as either a possible Uncorrectable Error (UE) or malicious data alteration simply because of a stale parity. Unfortunately, NVMs have a very high Random Bit Error Rate (RBER), ranging from $1\times10^{-3}$ to $1\times10^{-6}$ [66]. The probability of having at least one single-bit upset after a system crash (for the small set of data cache lines that can possibly have their parity lost in a crash) can be as high as 99% in PCM-based non-volatile memory for a small 32kB parity cache. Although the system may have lightweight in-DIMM error correction to offset some random errors, it will still be insufficient. For instance, DDR5 specifies in-DIMM single-bit correction per 128b data [2]. We observe that even such in-DIMM correction is insufficient to prevent recovery failure caused by errors. Therefore, it is highly likely that such a coordinated security-reliability feature with caching will render the system 'Not Securely Recoverable' after a system crash—meaning that we will have to either compromise the security or system reliability, once a system crash occurs. Consequently, the coordinated implementation of Security and Reliability by using MAC's error detection capability is not currently suitable for NVM.

In this paper, we re-architect such a coordinated security-reliability approach for NVMs. In particular, we revisit two fundamental aspects of such a method to make it suitable for NVM. First, we rethink the position of both MAC and parity and identify which one is cachable and recoverable at the same time. Second, we design our scheme such that it is capable of tolerating both random errors as well as chip failure during both run-time and recovery time.

**Contributions.** We make the following contributions:

1) We propose **R**ecoverable and **C**hipkill capable NVM, RC-NVM, the first recovery-aware implementation of security-reliability co-operation in NVM that is capable of maintaining chipkill level protection not only during run-time but also after a system crash.
2) We identify that it is better to co-locate parity/ECC with the data and cache MACs in a recovery-aware implementation of security-reliability cooperation for NVMs. We integrate the MACs into the Merkle-tree to be able to recover them after a system crash.
3) We co-locate Reed-Solomon code instead of parity which is capable of correcting random symbols, as well as an entire chip with the help of the MAC.

We evaluate our scheme in the Gem5 [12] simulator (open source version) and analyzed its performance with several memory-intensive workloads from SPEC® 2006 CPU, Whisper [36], and in-house persistent memory benchmarks. Our proposed solution allows 1.18x speedup and 27% write reduction compared to the state-of-the-art.

**Organization.** We present the background (including the threat model and key motivation) in section 2, the design in section 3, our simulation methodology in section 4, the evaluation results in section 5, the discussion in section 6, the related work in section 7, and the conclusion in section 8.

## 2 Background and Motivation

### 2.1 Threat and Fault Models

#### 2.1.1 Threat Model

Similar to the state-of-the-art NVM Security works [10], [14], [41], [51], [58], [60], [71], [74], anything stored outside the processor chip is considered vulnerable in our threat model. The processor chip is assumed to be secure, trusted, and difficult to tamper with. The attacker can tamper with the memory contents, and maliciously modify or replay data or security metadata stored in memory. The attacker can also attempt to snoop the data bus to read processor-memory communication. The attacks that physically tamper with the processor chip to extract secret information, such as encryption key and root hash are excluded from our threat model. Leakage through access pattern is also orthogonal to our work and not part of the threat model.

#### 2.1.2 Memory Organization and Fault Model

Typically, bits in memory are grouped together in multiple banks that form a memory chip. A group of such memory chips is aligned to form a memory rank in the Dual In-line

Memory Module. Memory access is performed by activating all chips in a rank, where each of the chips supplies some portion of the accessed data. Typically, the size of the cache line is 64B, and each of the memory chips supplies some bytes. For instance, in an 8x8 DIMM, each chip will provide access to 8B of the data. We assume NVDIMMs to have similar organization and access behavior. While our work is not restricted to any particular form factor or memory organization, we assume x8 ECC chips where a rank has eight data chips and one ECC chip. We also assume Phase Change Memory (PCM) as the underlying technology for NVM. However, our work can be seamlessly integrated with any non-volatile main memory technology (e.g., ReRAM, STT-RAM). Due to PCM's popularity as a main memory technology in available and commercial NVDIMMs [22], we primarily focus on this type of technology.

While the memory organization in NVM can be similar to DRAM, the fault model is different. In DRAM, the majority of the faults are permanent in nature (approximately 70% [48]) and can be masked during production using row or column sparring. Transient fault can also occur in DRAM due to radiation and particle strikes [49]. Although PCM cells are not vulnerable to particle strikes, they can deviate from their original state due to resistant drift and read disturb [39]. Since NVM cells are not refreshed often as DRAM, transient errors may occur more frequently, and get accumulated over time, and that is a significant concern for reliability [39], [66]. Hence, they require more robust protection against random bit upsets. Server memories are generally equipped to tolerate an entire chip failure. However, a chip failure can result from many internal upsets, including failure in the memory circuitry and failure of a large group of bits. Consequently, when designing reliable NVM systems, we must consider both random errors and chip failures. Therefore, similar to prior works on NVM reliability [66], [72], we consider random failures in multiple chips and complete chip failure.

## 2.2 Encryption, Protection, and Consistency

### 2.2.1 Memory Encryption

Counter Mode Encryption (CME) has been widely used in the literature [44] and industry implementations [17] for ensuring the confidentiality of data stored in main memory. Simple encryption mechanisms(e.g., Electronic CodeBook, or ECB) are not secure as they cannot preserve the spatial and temporal uniqueness of the encrypted data. In memory encryption, it is important to prevent the attacker from inferring data by analyzing the temporal and spatial relations between the encrypted cache lines. In the main memory implementation of CME, every memory block (usually 64 Byte) is given a dedicated counter. These counters can be considered as the version numbers of the data blocks, which are updated on every memory write.

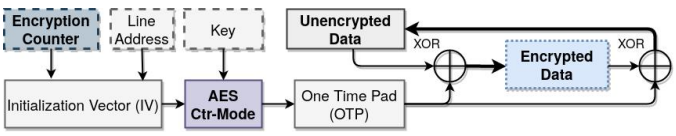


Fig. 1: Counter Mode Encryption.

Figure 1 shows a basic operation of the CME. During encryption, an Initialization Vector (IV) is generated using an incremented encryption counter and the address. This 'IV' is then encrypted using a key to generate One Time Pad (OTP). Incrementing the counter before OTP creation guarantees temporal uniqueness of memory writes, which is critical for memory encryption to be secure. Additionally, the use of the physical address of the data to generate OTP also ensures that the same data will be encrypted with different OTP if they are from different locations. To encrypt the data, OTP is XORed with the data, and the encrypted cache line (cipher) is written to the memory. The decryption operation is also similar to encryption; the OTP is created using the counter, address, and key, which is XORed with the ciphertext read from memory to decrypt the data. Note that the generation of OTP and reading the cipher from memory can be done in parallel, which effectively hides the decryption latency.

### 2.2.2 Data Tampering and Replay Protection

Generally, the integrity of the protected data is ensured through a cryptographic hash value calculated over the data. In a secure memory architecture, a hash tree (popularly known as the Merkle tree) is used for this purpose. Such a tree can be generated by hashing the data contents at multiple levels, which eventually collapses into one single hash (the root of the tree). The root hash consists of several hashes (in case of non-parallel BMT) or counters (in case of SGX-tree) as shown in figure 2, which can be securely stored within the processor chip. The size of the tree depends on the size of the protected memory, and the arity (`n-ary` means each hash node has `n` children) of the tree. Therefore, protecting a large memory requires an extremely large tree size. Prior works [44] have proposed ways to reduce the size of the tree by building the tree only over the encryption counters, not the entire memory. Such a design is known as Bonsai Merkle Tree (BMT), where the data blocks are protected using a Message Authentication Code (MAC) calculated over the data, address, and counter. Figure 2 illustrates such a tree. While this type of tree has the advantage of simplicity and authenticating the protected contents using a single root, its update is sequential (non-parallel). As used in Intel's SGX, there is also a parallel style BMT where the intermediate nodes accommodate 'version counters' and MAC instead of hash values. In this way, it is possible to update all respective counters in a branch and calculate new MACs for all nodes in that branch in parallel (Figure 2b). Note that we aim to recover *Data MACs*, and is orthogonal to the choice of the tree structure.

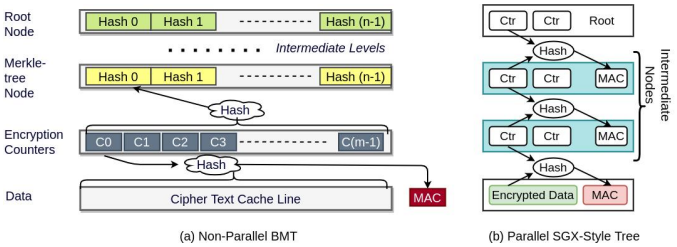


Fig. 2: Merkle tree organizations.

**Synergy in NVM.** Synergy [45] observes that we can use the extra bandwidth to co-locate data MAC with its data if ECC-DIMM is used. This design allows fetching the {data, MAC} pair in a single memory read operation. If the fetched data passes the MAC verification, it can be safely assumed that no malicious (attack) or non-malicious (error) data modification has occurred. Synergy also stores parities calculated over the data chips in a separate location to preserve the correction capability. If the MAC value calculated using the counter, data, and address does not match the co-located MAC that is fetched with the data, Synergy first assumes that this is a possible chip failure. In such an event, Synergy speculatively corrects the faulty chip using parity and verifies the correctness with the MAC. Assuming the parity bits are not faulty, one of eight trials will successfully correct the chip. While this is a simple and efficient method to achieve higher reliability and improved run-time performance in a secure memory system, it has to update parity in the same way MACs are updated in the original BMT proposal to preserve such error correction capability. Figure 3 shows Synergy's memory organization compared to the original ECC-DIMM.

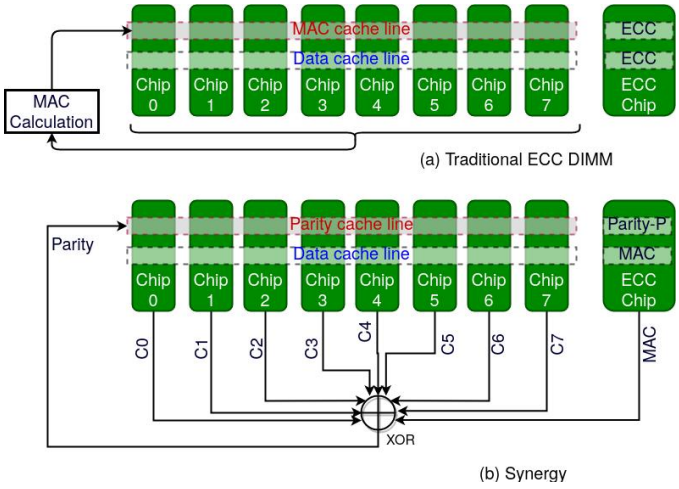


Fig. 3: Synergy's data layout (b) vs. typical data layout in ECC DIMM (a).

### 2.2.3 Metadata Crash Consistency

Although caching security metadata in a fast and volatile cache improves the performance significantly, it can leave the system in an inconsistent state if a sudden system crash happens. The volatile cache holds many dirty and updated metadata, which are usually later reflected in the memory when they are evicted from the cache. Therefore, the memory version of the metadata can be stale if the volatile contents within the cache are erased due to a system crash before they are evicted.

Based on the prior works related to security metadata crash recovery, the recovery efforts can be generalized in three categories: (i) encryption counter recovery, (ii) recovery of hash nodes in non-parallel BMT, (iii) version counter recovery in parallel BMT. Among the security metadata, the encryption counters have a distinct incremental property that allows the lost value to be a few increments away from the stale value found in the memory after a crash. This property allows trial-and-error of the encryption counters if they can be sanity checked by another means.

For instance, Osiris [60] achieves this sanity check using encrypted ECC, and CC-NVM achieves this by using data MAC. Incrementing a counter and checking its sanity using encrypted ECC, or MAC can reveal the original value that was lost in a crash. Once the counters are repaired, the hash nodes in non-parallel BMT can be easily re-calculated and verified against the root of the tree. While (i) and (ii) can be easily done, (iii) can be challenging since the version counter values can neither be simply re-calculated from the lower level nor recovered using the trial-and-error method as there is no available sanity checking method for them. In such a case, Anubis persistently tracks the LSBs of the version counters in a shadow memory region and uses lazy style tree update[2] to limit the updates in the tree from tree level $n$, to only 1. While this provides recoverability of parallel BMT nodes, it still incurs 2x writes. This is solved in Phoenix [5] and STAR [20], where the updates in the bottom level in the tree (encryption counter) are not persisted strictly. Instead, Phoenix uses a sanity check mechanism to recover the bottom level and persistently tracks the upper levels. STAR persists the LSBs with the data MAC assuming that the MAC size is 56b, similar to the SGX-tree proposal [14], and some unused bits are available. Unlike encryption counters, hash nodes, or version counter nodes, the parity or data MAC cannot be recovered in such a way. They cannot be simply incremented and sanity checked like encryption counters, making [58], [60] ineffective. Even if we assume that some unused bits are available similar to Intel's SGX tree where we can store the LSBs of the parity or data MAC, it will not be effective. There is no guarantee that MSBs will not change in the next update. Indeed, such an event is highly likely in the newly calculated parity or data MAC. The only option is to persist the updated parity or data MACs on every data write, which takes the solution right where the problem began.

## 2.3 Performance Issues

### 2.3.1 Recovery Time Problem

Recently explored secure NVM architectures [10], [71] show that it is not only the recoverability but also the high recovery time is a major challenge in adopting large scale secure NVM system. Anubis [71] shows that it could take hours to recover a secure NVM protected using Counter Mode Encryption and Merkle tree. The main reason for such a long recovery time is simply because the system cannot pinpoint the stale metadata after a system crash, requiring aggressive scanning and multiple trials to attempt metadata recovery. To allow faster recovery, Anubis maintains a shadow region in NVM and tracks the address of security metadata whenever the memory controller modifies them inside the cache. During the entire life cycle of security metadata inside the cache, from the placement to the eviction, it only needs to be tracked once when it is modified for the first time. As such,

2. Unlike eagerly updated tree where all nodes in a branch are updated up to the root, lazy update only updates the lower level. Whenever a node from the tree is evicted, the upper level is updated. This lazy propagation makes the root of the tree stale and hence depends on a secondary root as used in [71] and [5].

the system can pinpoint the stale metadata and securely recover them after a system crash.

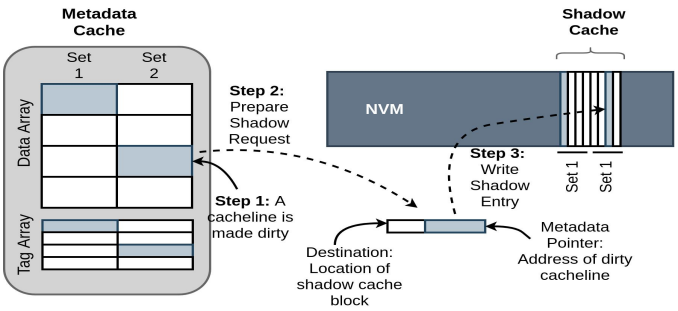


Fig. 4: A simple shadow caching example.

A shadow tracking mechanism for a cache structure is shown in Figure 4. The idea is to allow NVM to mirror the state of the cache, where the gray box in the figure represents the dirty blocks. The size and organization of the Shadow Cache are identical to those of the cache it is mirroring. If a cache line is marked dirty for the first time (step 1), a shadow request is prepared. The destination address of the request is prepared based on the respective location of the dirty cache line in the cache (step 2). The data content of the request holds the actual physical address of the cache line that has become dirty inside the cache and is being tracked. Finally, the shadow structure in the memory is updated. The atomicity of the write requests (data, metadata, and shadow write) must be preserved. Note that the subsequent update of a dirty block does not need further tracking. When evicted, the corresponding entry in the shadow region does need to be erased. This relaxed tracking allows a reduced overhead using a few trials during recovery. In other words, a stale shadow log will not harm recovery. If any new metadata transitions from clean to dirty in the same position, the stale log is updated with the new one.

#### 2.3.2 *Preserving Atomicity of Writes*

Modern architecture guarantees that any write which makes its way to the NVDIMM is persisted. However, in an ideal threat model, where only the processor chip is trusted, security implementations need to be within the trusted processor. In such an architecture, each memory write can issue additional metadata writes–all of which may not be committed to NVM at the same time. For instance, evictions from the metadata cache (counter, hash node, or MAC) and shadow entry update requests are generated when serving a data write request. These write requests are placed in the write buffer before finally being updated in NVM. Atomicity of these writes must be guaranteed so that a batch of data and security metadata updates is committed at the same time. Instructions like `pcommit` can be used to flush the Write Pending Queue (WPQ) to maintain atomicity. Modern processors have started to provide hardware support [1] for atomicity and persistence of the Write Pending Queue contents. For instance, Intel supports the Asynchronous DRAM Refresh (ADR) feature [24], which ensures that the contents loaded in the Write Pending Queue (WPQ) are flushed on a power loss event. However, the size of the WPQ is extremely limited (only tens of entries) due to the limited backup from ADR. State-of-the-art memory security and persistence models utilize this limited ADR support to guarantee that a batch of the atomic write requests is atomically written. The cipher, evicted counter, tree nodes, MACs, and shadow requests are temporarily stored in some persistent registers. Next, they are moved to WPQ and erased from the registers once all writes are moved into WPQ, which is backed by ADR support. If power fails before pushing all writes to WPQ, another attempt after the system reboot will be taken to persist the batch. The requests that fail to reach the persistent registers will not be committed.

### 2.4 Motivation

Reliability-Security coordination in secure NVM can be done using the error detection capability of data MAC, similar to many prior approaches that have achieved such for DRAM. However, to make it practically usable in NVM, we need to cache the parities (if MAC is co-located with the data), or MACs (if parity is co-located with the data). While caching improves performance and NVM's lifetime, it creates a single point of failure even with a single bit error after a system crash. If parities are cached while MACs are persisted with the data, the system may identify an error using the MAC but lost parities will prevent reliable system recovery. On the other hand, if MACs are cached, post-crash data verification will not be possible.

Figure 5 illustrates such a scenario by varying the cache size. The Y axis in the graph represents the probability of having at least one error out of all data bits that are likely to lose their MAC or parity after a system crash. Here we assume each cache line in the cache will be able to hold eight MACs or parities. Therefore, a single-bit upset in $8 \times S$ kB memory data can possibly create a single point of failure, where $S$ is the size of the MAC/Parity cache. Here we assume that the memory cells have certain bit error rate and SEC in-DIMM ECC for each 128b is available similar to DDR5 [2]. While this analysis is done purely based on Raw Bit Error Rate (RBER) of the NVM cells for generalization and keeping the analysis independent of specific technology, there can be many reasons for post-crash data errors. Glitches in the memory circuitry caused by a power failure can also lead to failure in larger granularity (e.g., entire row, column, bank, or even entire chip) [68]. To make caching effective in terms of performance, its size needs to be reasonably large. As it is clear from Figure 5, a reasonably large cache size (e.g., greater than 64kB) will have recovery failure probability more than 50% in most cases. Beyond 100kB, the failure is almost certain for average NVMs even with in-DIMM ECC. Clearly, additional support to securely recover the system is needed in such reliability-security coordinated architectures.

## 3 RC-NVM

This section discusses the design options and challenges in a MAC-assisted reliability scheme suitable for NVM.

### 3.1 Caching MAC vs. Parity

In MAC-assisted memory reliability, we can either co-locate the MAC or the parity information in the ninth chip. If the MAC is co-located with the data, the parity is stored

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3279031



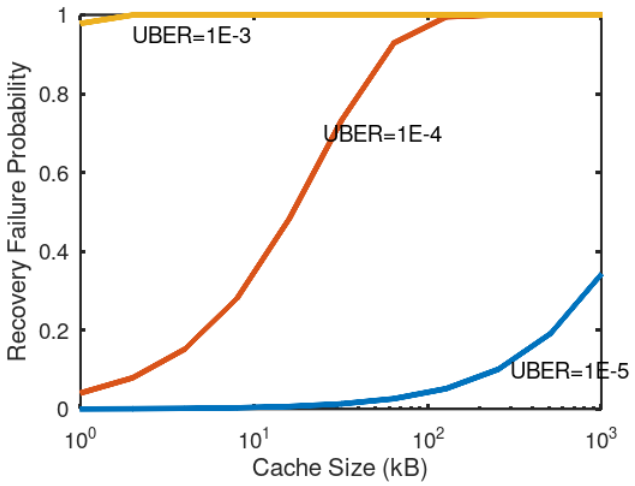


Fig. 5: System failure probability due to post-crash data error.

in a dedicated data memory region and vice versa. As originally proposed in Synergy, the MAC is co-located with the data, and the parity is stored separately. The primary motivation to co-locate the MAC with the data for such a DRAM-based scheme is to allow checking for both errors and malicious modifications at the same time, without any additional reads. If there is no MAC mismatch, it can be safely assumed that there is no error in the data, hence no need to read parity. Assuming that the error rate of the memory technology being used is extremely low, the data will be simply errorless in most occurrences, and hence no correction is needed. Therefore, most of the memory reads will be low overhead as costly trials using parity for error correction can be avoided. This can be true in the case of a DRAM having low Raw Bit Error Rate ($\approx 1\times10^{-7}$), where the cells are also frequently refreshed. However, it can trigger frequent error correction trials[3] in PCM-based Non-Volatile memories due to high RBER, and lack of cell refresh. With the MAC-assisted reliability being the primary defense, the MAC latency can be tens of nanoseconds (e.g., 40ns [15], [32]) and hence can cause a significant slowdown for frequent error trials. Clearly, triggering such high-overhead trial-and-error-based chip correction simply because of a few bits of error is not the best approach for dense NVMs. The better solution will be to employ a comparatively low-cost error detection and correction method that can filter out some random errors before even calculating the MAC.

For the above reasons, we use Reed-Solomon code instead of plain parity, which is faster compared to MAC-based error detection. RS code is able to detect a number of random errors and correct a subset of them. The MAC can then be used as a second line of defense against larger granularity error (e.g., chip failure), or Silent Data Corruption (SDC) that the weaker ECC missed in the beginning. This will preserve the costly trials with repeated MAC calculations only where and when they are required. This is challenging in the context of MAC-assisted error correction without extending the size of the ECC. Fortunately, Reed Solomon (RS) codes have the capability to act as an erasure parity if the location of the error is known and also correct a limited number of random symbol errors if the location is unknown [30], [66]. A $2t$ symbol long RS ECC code is capable of correcting all combinations of random $v$ symbol errors, and $e$ symbol erasures given that the inequality $v + \frac{e}{2} \leq t$ holds true [30]. This means that we can correct $t$ random symbols if no erasures are present and $2t$ erasures if no random symbol errors are present. An $s$-bit symbol can provide coverage to $2^s - 1$ symbols. Hence, the minimum reasonable symbol size with an 8B RS code will be 8b. Such code will correct up to four random erroneous bytes without the help of the MAC and eight erasure bytes (equivalent to an entire chip in x8 DIMMs) if MAC is used to identify the faulty chip. The 8B RS code can be co-located with the data and fetched easily while reading data, as originally done in ECC DIMMs. On the other hand, the MAC can be stored in a dedicated memory region and cached inside the processor to avoid frequent memory reads for MACs.

**Observation:** Triggering costly trials for chip correction in a MAC-assisted reliability scheme is high overhead, which can be extremely inefficient in dense PCM-based memories. A reasonable solution is to use co-located RS ECC instead of parity as the first line of defense and store MACs separately while caching them.

Intuitively, one can argue that we can still co-locate the MAC with the data and simply cache the RS codes within the processor. We can still apply the RS bytes initially to filter out random errors before checking with the MAC. However, this is an odd choice of placement given that the first operation after reading the data will be correcting random symbol errors, not integrity verification. While the order of error checking and MAC verification is one reason to co-locate RS-ECC with the data, it is clearly preferable if the error correction during recovery time comes into the picture. If we lose the parity or RS bytes, we lose correction capability for some data and hence the ability to recover from a crash. On the other hand, if we lose MACs, they can be recovered if we preserve the data correction capability through co-located RS bytes across crashes and employ a secondary verification method to determine the authenticity of the recovered MACs. Note that, we assume that the MACs can be recovered for now; the details of the recovery procedures using a secondary verification method are described in Section 3.4. This secondary verification is a fundamental aspect of RC-NVM's recovery mechanism and will be discussed later. Thus, co-locating the RS ECC bytes (to preserve error correction capability) with the data and caching MACs is a more reasonable and practical choice for recovery-aware NVMs.

**Memory Organization.** Since the reason to allow MAC caching and co-location of RS ECC with the data is clear from the above discussion, we now focus on illustrating the details of the system organization and recovery mechanism.

Figure 6 shows our data, ECC, and MAC organization. Each data chip (D0-D7) stores eight bytes of data, and the ECC chip stores eight bytes RS ECC calculated over the entire 64B cacheline. Similarly, a 64b MAC is calculated over the 64B data cache line, and initially written to the dedicated

3. Synergy requires at least eight MAC calculation for an error correction, which can be 16 in the worst case if parity is also erroneous.

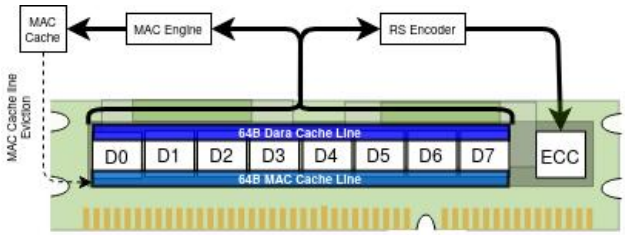


Fig. 6: Organization of MAC, RS ECC, and DATA in NVM.

MAC cache. A MAC cache line, consisting of eight MACs, is only written back to the NVM if it is evicted from the cache.

### 3.2 Run-time Data Error Correction

Once data and its corresponding RS ECC bytes are read from the memory, the data is checked for errors using the ECC bytes. The error checking in Reed Solomon code involves encoding to re-calculate the parity information and syndrome generation—The syndrome is simply the vector sum of the received parity code and recomputed parity code [30]. Encoding and error detection using RS code is comparatively faster than decoding (e.g., 1.6ns for encoding vs. 45ns for decoding [66]). Decoding is only needed when errors are identified after syndrome calculation and corrections are required. The RS ECC bytes will be able to correct up to four bytes of random symbol errors (32 bits total) before passing the data to the MAC calculation engine. However, if the number of errors in the data spans more than four bytes, the ECC may start miscorrection or silent error propagation. This will be identified when the MAC calculation is done over the corrected data and is matched against the previously calculated value. A mismatch between the calculated and pre-stored MAC will indicate one of the following events: (i) RS ECC has miscorrected the data or silently propagated the error, (ii) An attacker has maliciously modified the data.

Whenever such an event occurs, the memory controller optimistically assumes that a chip failed and starts correcting it using MAC trials. During such a trial, the originally fetched data is used as input since miscorrection by the RS ECC will likely expand the error beyond one chip and the recovery will fail. In each trial, one of the chips is assumed to have failed and then corrected by using RS ECC bytes as erasure code. After correction, the MAC validation indicates whether it was a successful correction or a failure. A match in the calculated and pre-stored MAC means that the faulty chip has been corrected successfully. If none of the trials is successful, it may indicate multi-chip failure, five random byte failures that are not enclosed within a chip, or a malicious modification of the data.

### 3.3 Root-Connected MAC for recovery

In a traditional Bonsai Merkle-Tree, the tree is generated on top of the encryption counters, and data MACs are stored separately (Figure 2). While the leaves of the tree represent the encryption counters, the MACs are separated and are connected to the root indirectly through the encryption counters. This organization makes it difficult to replay a MAC or data block without replaying the counter (which is detectable using the root). While this guarantees that data alteration will be detected through the MAC and MT root during run-time operation, recovery-time data verification can be challenging if updated MACs are lost from the volatile cache. Assuming that we can recover the counters using the available methods, we can attempt to recompute the MAC using the data, address, and counter. However, there is no way to verify whether the newly obtained MAC is authentic since it also depends on the data. The attacker may tamper with the data blocks that correspond to the lost MACs, and there will be no way to detect this. While this is the fundamental reason why post-crash data verification is challenging if we cache MACs in a cache, this is also the reason why we cannot recover the lost MACs.

One way to circumvent this problem is to avoid using the BMT structure altogether and create a larger Merkle-tree over the data instead of the counters. While this would eliminate the necessity of using data MACs and, hence, recovery, such a design would have a larger Merkle-tree and increase both performance and storage overheads for the Merkle-tree verification. Similarly, an alternative design where the MT is built over the MACs instead of the counters would have similar issues. On the flip side, if there were a way to verify the correctness of the recomputed MAC in BMT, we could easily recover the MAC after a system crash. This would prevent any post-crash tampering of the data blocks that correspond to the affected MACs.

**Observation:** Lack of direct connection between the root of Merkle-tree and MAC prevents post-crash data verification and MAC recovery if MACs are cached in a volatile cache.

It is possible to authenticate a recalculated MAC after a system crash if we include the MACs as additional inputs to compute the Merkle-tree root. The number of children of a node determines the arity. For instance, figure 7 shows a naive way to include MACs into the original Merkle-tree in a system that uses 64-ary split counter organization [56] and 8-ary Bonsai Merkle-tree above the counters. Each 64B cipher block is associated with a 56b MAC. A node in the lowest level of the Merkle tree covers 512 cipher blocks. In this organization, MAC and Merkel-tree hash size is 56b, similar to Intel-SGX. Therefore, intermediate MT nodes can store an additional 56b data in the unused space. To integrate the MACs into the main tree and hence make a connection between the MACs and the MT root, we can hash MACs of the 512 consecutive ciphers in the physical address space into a single 56b hash which is stored alongside the hash of counters in the lowest level of the Merkle-tree. The 512b node in level 1 that accommodates eight counter hashes and one MAC hash will be hashed into a 56b value in the upper level, the MACs will be used for the root calculation. The MAC hash calculation can be done similarly to Merkle-tree calculation by hashing MACs in several levels. In this way, intermediate hashes for the MAC hash calculation can be temporarily stored in memory to reduce the number of hash calculations.

With this modification in the Merkle-tree organization, reading from memory will be unchanged. The 56b MAC Hash in the lowest level of the Merkle tree is not needed for

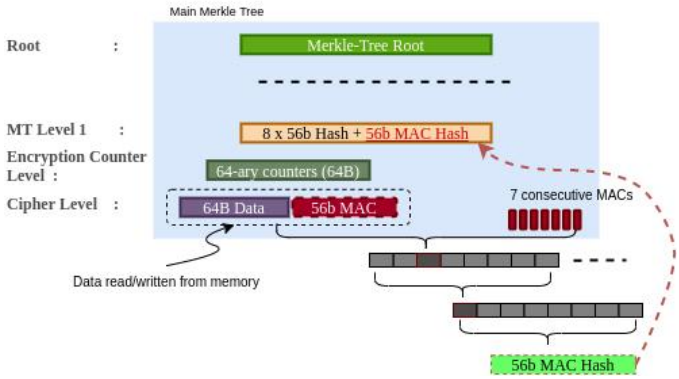


Fig. 7: Naive way of implementing root-connecting MACs.

data verification during run-time and hence ignored during reads. The integrity of the ciphers read from memory is verified using the associated MACs, and the counters are verified using the Merkle tree as done typically. However, during a write process, MAC hashing needs to be done, and the calculation of level 2 will have to stall until the 56b MAC hash is available. Assuming a duplicate hash engine is available, MT Level 1 and MAC Hash calculation can start in parallel. While simple, this approach has several drawbacks. First, all input MACs may not be present in the cache and will incur several additional memory reads. This will incur significant stalls during write operations. If we organize this hashing process in a Merkle-tree structure and store the intermediate nodes (colored in gray), we can possibly reduce the number of MACs we need to read for such calculation. More precisely, for 512 MACs in the lowest level, we need a four-level Merkle-tree and hence need to read three cache lines to calculate the MAC; one containing eight MACs (one new MAC and seven other neighboring MACs) and two upper-level nodes. However, in terms of performance and storage overhead, this solution behaves similarly to a naive Merkle-tree structure that is built over the data only. Consequently, there is no justification for using such a design if we use BMT. The primary problem here is due to the fact that the MAC hash calculation process in this design is non-incremental, and we need additional inputs. In other words, it is not possible to update the MAC hash only using the updated MAC (colored in red), avoiding all other neighboring and intermediate MACs.

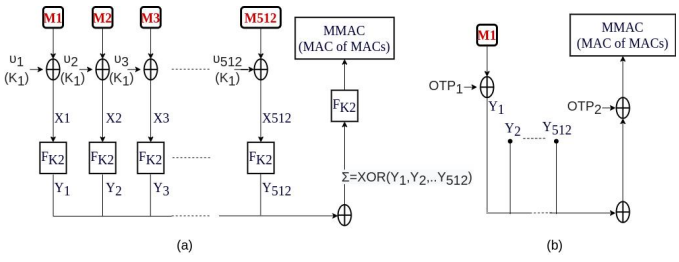


Fig. 8: Calculating MAC of MACs in an incremental fashion.

To reduce the overheads of the aforementioned design, we utilize the concept of Parallel MAC, or PMAC [42], [43]. The PMAC construct allows the calculation of a unique MAC over many inputs in an incremental fashion. Figure 8a shows how a single MAC can be calculated over 512 MACs in this fashion. The first step in this process is to XOR the individual inputs (MACs) with a unique value ($\upsilon_i$) that is dependent on a secret key $K_1$. Next, the XORed Values ($X_i$) are then scrambled using a block cipher F that takes another secret key $K_2$. The results ($Y_i$) are XORed all together into a single value $\Sigma$. Finally, the $\Sigma$ is fed into $F_{K2}$ to get the final MAC. Here, we refer to this final MAC as the MAC of MACs or MMAC. We can use this MMAC in place of the MAC Hash in the prior design. To be able to perform an incremental update, we should be able to invert MMAC and get the $\Sigma$ back. One advantage of calculating MMAC in this way is that we can re-use the AES engine for the function F. Since, AES-CTR mode XORs the pad in the final stage, it is possible to get $\Sigma_{i-1}$ back if we XOR the MMAC with the pad we have used to encrypt it. To update the MMAC with just a single MAC, we need to recover $\Sigma_{i-1}$, and perform $(\Sigma_{i-1} \oplus Y_i \oplus Y_{i-1})$ where $\Sigma_{i-1}$ is the previously calculated MMAC fetched from the MT node, $Y_i$ is the output from $F_{K2}$ for updated MAC, and $Y_{i-1}$ is the output from $F_{K2}$ for previously calculated MAC for the same data block. We can leverage the uniqueness of the PAD in AES-CTR to combine the first two steps into one by XORing the MAC with a truncated version of the OTP created using AES-CTR ($OTP_1$, as shown in Figure 8b). Finally, all Y values are XORed and encrypted using another PAD calculated using a different key. As soon as the AES engines finish the data encryption process, we can re-employ them to create four OTP—generally, four 128-bit AES engines are used to encrypt 512-bit of data. After encryption, these engines sit idle, and we use four of them to calculate the MMAC; $OTP_{1i}$, $OTP_{1(i-1)}$, $OTP_{2j}$, and $OTP_{2(j-1)}$. Since AES encryption finishes before the MAC calculation [34], we can start calculating $OTP_2$ even before the MAC calculation finishes. Once we have the new MAC $M_i$, old MAC $M_{i-1}$, old MMAC and two OTPs, we can quickly perform $\Sigma_j = (\Sigma_{j-1} \oplus Y_i \oplus Y_{i-1})$ which is XORed with the $OTP_{2j}$ to get the updated MMAC. This process does not have any additional storage overheads and incurs small delays in MT updates.

### 3.4 Recovery Procedure

After a system crash, the volatile contents from the SRAM caches are erased. This includes all cached encryption counters, Merkle-tree hashes, and the MACs. With shadow tracking of security metadata similar to prior works [5], [71], it is possible to pinpoint the stale metadata.

First, the encryption counters are recovered using any of the state-of-the-art recovery methods [31], [60], [71]. For instance, to recover the encryption counters, we can try to decrypt the data with the stale value of the counter and calculate the RS Syndrome. If a wrong counter has been used, or if there are some errors present in the data, it will likely produce a non-zero syndrome. The recovery engine first speculatively tries all the possible counters assuming there is no error in the data. The number of trials for each counter can be limited by setting a threshold value for updates of a counter inside the cache, similar to Osiris [60]. If all counters produce a non-zero syndrome, the reason is likely a data error. At this stage, the recovery unit starts treating the RS bytes as erasures and speculatively tries to fix

**Algorithm 1:** Crash Recovery Procedure

```
Read Shadow_Entries (Stale_Counters, Stale_MACs, Stale_MT);
#Counters
for all C_i in Stale_Counters do
    while C_ij= C_i+m do
        Decrypt Data; Calculate Syndrome;
        if non-zero syndrome then
            try_correct();
        else
            continue;
        end
    end
end
#MACs
for all M_i in Stale_MACs do
    Recalculate MACs;
end
#MT
for all MT_i in Stale_MTs do
    Recalculate MT Hash from children;
end
#Verify
for all C_i in Stale_Counters and all M_i in Stale_MACs do
    Traverse all the way up to root;
    if root_mismatch then
        Recovery Failure;
    else
        Continue;
    end
end
```

each chip's contents, and verifies again by recalculating the syndrome. This way, an entire failed chip can be tolerated. Note that this process is also done for all possible counter values. If there are $n$ chips and $m$ possible counter values (e.g., four in Osiris), the total number of trials in the presence of errors for each block will be $m \times n$. After the recovery of the counters, we speculatively recalculated the MAC using the data fetched from NVM and the encryption counter. This is followed by an attempt to verify the data by traversing all the way up to the root. Since the Merkle-tree covers both the counters and the MACs, any tampering with the data will be detected. Algorithm 1 shows our recovery procedure in brief.

## 4 Experimental Methodology

We evaluate our RC-NVM's performance using the Gem5 [12] simulator. We use a mix of persistent and non-persistent workloads. Note that we use the NVM to host both persistent and non-persistent data. Since the security metadata is managed in the memory controller and transparent to the software, the operations undertaken in the memory controller will be the same regardless of what application is being used. We have used SPEC CPU 2006 benchmarks [3] for non-persistent workloads and a mix of in-house benchmarks and benchmarks from Whisper [36] as persistent workloads. The in-house microbenchmark `uX` accesses one byte after every X byte in a sequential manner with a read/write ratio of 1. We checkpoint after the application's initialization phase for each application and simulate 500M instructions after that. We have implemented counter mode encryption, the Bonsai Merkle Tree, and MAC verification for the baseline system. A write-back cache has been integrated with the memory controller for caching encryption counters and Merkle tree nodes. Finally, we incorporate the RC-NVM system that caches MACs and performs necessary operations to enable MAC recovery. We varied the cache size from very small (8kB) to large (512kB) and analyzed the cache performance, speedup, and recovery time for sensitivity analysis. Table 1 lists our parameters.

TABLE 1: Simulation Configuration Parameters

| | |
|---|---|
| **Processor** | |
| Core | 4 Cores, X86, OoO, 2.66GHz |
| L1 Cache | Private, 2 cycles, 32KB,8-Way |
| L2 Cache | Private, 20 Cycles, 512KB, 8-Way |
| LLC | Shared, 32 Cycles, 8MB, 64-Way |
| **PCM Main Memory** | |
| Size | 16 GB |
| PCM Latencies | Read latency 100ns, Write latency 300ns |
| **Encryption** | |
| Metadata Cache | 256kB, 8-way, 64B Block |
| AES latency | 40 cycles [32] |
| Encryption Counter | 64bit major, 7 bit minor Split Counter Organization |
| Merkle Tree | 8-ary Bonsai Merkle Tree |
| MAC cache | 64kB, 4-way, 64B Block |
| MAC Latency | 40ns [16], [32] |
| MAC Organization | 8 MACs/ Cacheline |

## 5 Evaluation Results

The simulated performance, memory access, and sensitivity results are presented in this section. In the rest of this section, we compare the following designs:

1) **Synergy-NVM (Baseline).** This scheme is similar to the Synergy [45] scheme in terms of MAC management. No extra reads or writes are required for MACs; however, each write needs to update the parity. Additionally, CME and Merkle tree are also implemented to ensure data confidentiality and integrity guarantee.
2) **Atomic Non Co-Located MAC.** In this scheme, MACs are written to a separate and dedicated memory region. For each memory read/write operation, MAC for the data is read or written atomically.
3) **Write-Through (WT).** This scheme uses a write-through cache for MACs, where MAC reads are satisfied from cache but writes must be written back to the memory.
4) **RCNVM.** Our proposed scheme, RCNVM, which uses a write-back cache for MACs and allows MAC recovery through integrating MACs into the Bonsai Merkle Tree.

### 5.1 Impact on Performance

Figure 9 shows the normalized speedup for different schemes compared with the baseline Synergy. As it can clearly be observed, data MACs in secure memory architecture can severely hamper the system's performance in Atomic Non-Co-Located MAC scheme, where MACs need to be written to and read from memory for every data write read. Our evaluation shows that such scheme can incur approximately 15% (up to 25% in leslie3d) slowdown in comparison with the baseline Synergy. The performance of a write-through cache scheme is very similar to the baseline Synergy. The only downside of the WT scheme in comparison with Synergy is that the WT scheme will have to read a few additional MACs when cache misses for MACs occur on reads. Both Synergy and WT schemes have high overheads due to the extremely high number of writes for MACs and Parities in WT and Synergy, respectively. On average, the WT scheme is only 2.1% slower than the

This article has been accepted for publication in IEEE Transactions on Dependable and Secure Computing. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2023.3279031



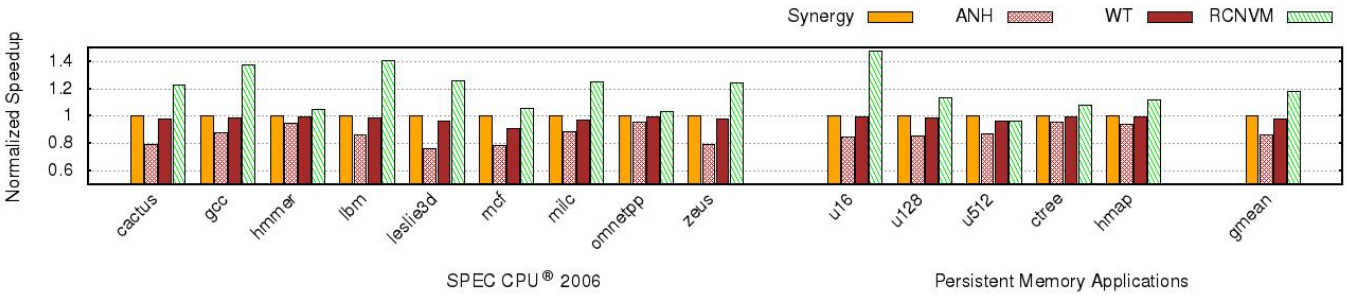


Fig. 9: Normalized Performance Improvement

baseline Synergy. On the other hand, RC-NVM uses a write-back cache that can allow dirty MACs to reside in cache, reduce the number of writes, and allow for crash recovery by calculating MAC over each consecutive 512 MACs and using it in Bonsai Merkle-Tree construction. On average, RC-NVM improves the performance by 18.2%. As shown in the figure, RC-NVM is the only recovery-aware scheme that can outperform Synergy in terms of performance.

## 5.2 Memory Access Overheads

### 5.2.1 Writes

Figure 10 shows the writes percentage of RC-NVM scheme compared to Synergy, where a significant reduction is achieved by RC-NVM scheme. Although Synergy interchanges the MAC and ECC position and has no extra memory access for MACs, it still incurs a high number of writes for updating ECCs/Parities similar to the ANH and WT schemes. Therefore, the number of writes is the same in Synergy, ANH, and WT schemes; the only scheme that can offer reduced writes is RC-NVM due to the use of a write-back cache that coalesces writes sent to memory. The figure shows how many writes RC-NVM can save in comparison with Synergy/ANH/WT. As shown in the figure, RC-NVM can reduce almost 27% of the memory writes. Such reduction in writes allows both performance improvement and an extended lifetime for a secure NVM system.

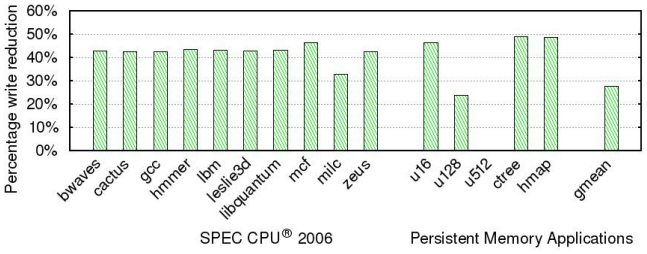


Fig. 10: Percentage of write reduction in comparison with Synergy.

## 5.3 Sensitivity to MAC cache Size

To further analyze the benefits coming from using a write-back MAC cache with eight MACs per cache line (similar to Intel SGX's counter organization [17]), we vary the size of the cache from very small (1kB) to very large (512kB) and analyze the cache hit rate and speedup for selected memory-intensive workloads. We also analyze the recovery time with varying cache sizes.

### 5.3.1 Cache Performance

Figure 11 shows the increase in hit-rate with cache size. As shown in the figure, the MAC hit rate in most applications increases with the increase in the size of the MAC cache. We notice that most of the applications' hit rate reaches their maximum with a cache size of 64kB. For instance, LBM quickly saturates at 64kB and does not significantly increase the hit rate with larger caches. MCF shows slight improvement with larger MAC caches.

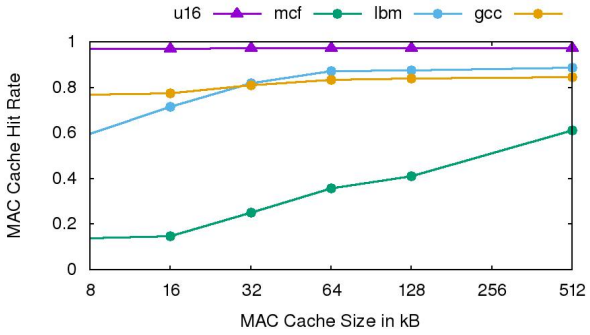


Fig. 11: Hit rate with varying cache size.

### 5.3.2 Performance Sensitivity

With a larger MAC cache, RC-NVM can reduce the miss rate and improve the performance significantly compared to Synergy. Figure 12 shows how speedup increases with the increase in the MAC cache size until a certain point, after which applications do not show any speedup with further cache size increase. After 64kB, most of the applications saturate in terms of speedup and only improve slightly with further increase in cache size.

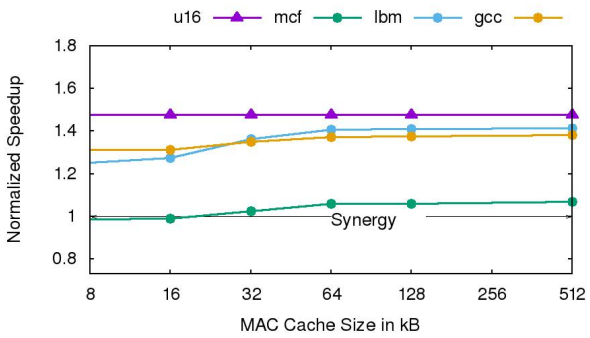


Fig. 12: Performance sensitivity to cache size

#### 5.3.3 Recovery Time Sensitivity

By varying the MAC cache size, we also analyze how long the system will take to recover from a power failure as shown in figure 13. For this analysis, we have assumed equal partitions of 128kB each for counters and Merkle tree in the metadata cache. Each counter cache line has 64 counters, and RC-NVM will have to fix all counters of a dirty cacheline. For simplicity, we assumed that all cache lines in the cache are dirty. During recovery, RC-NVM first recovers encryption counters. Then, it speculatively calculates MACs over the fetched data and recovered counters. The resulting MACs will be used for the verification of successful recovery by traversing the Merkle-tree up to the root.

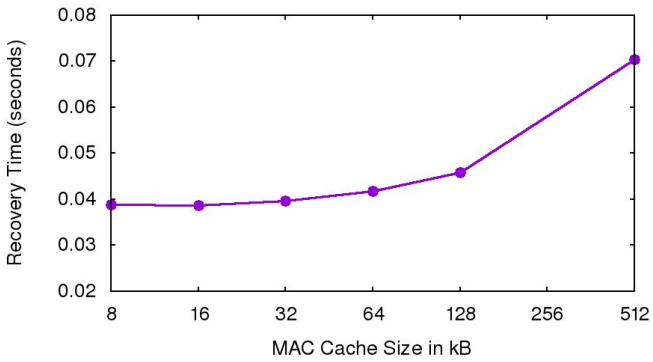


Fig. 13: Recovery time sensitivity to cache size

## 6 Discussion

**MAC Cache vs RS ECC Cache.** Although we assume that MACs are stored in a separate memory region and cached in a MAC Cache–an alternative design is also possible where RS ECC bytes are cached and MACs are co-located with the data. If RS-ECC bytes are cached similarly to ITESP [52], they can be re-calculated over the data if the MAC verification is successful. However, in case of a MAC mismatch, RS codes are needed to filter symbol errors and re-check by using MAC verification after error correction is done with RS codes. While both methods will have similarities in performance and reliability in current memory systems, allowing natural co-location of ECC with data is a more suitable choice for future NVM memory design for several reasons. The size of the MAC per 64B cache line is likely to remain unchanged for future memory systems as the MAC collision rate is not a direct function of DIMM organization or memory technology while the reliability is. On the other hand, the ECC/parity size varies depending on the memory design and the desired level of reliability.

Future memory systems will likely have higher error correction requirements and extended hardware support for stronger ECC. For instance, DDR5 supports 16-bits ECC per 64-bit data (divided into two channels) [2], [11]. Such a larger access granularity will render a 16B parity per 64B data cache line. Similarly, the parity/ECC size will also vary depending on the symbol size (e.g., 16B parity per cache line in a 4x16 DIMM for Synergy). It is impossible to use the larger access granularity of DDR5 to fit both MAC and ECC simultaneously in Synergy. For instance, if each channel in a DDR5 DIMM consists of 4 x8 data chips storing 64B data in 16 bursts, a chip will store 16B data of a cache line, and hence the entire 16B parity will be needed to provide Synergy's chipkill level protection. Storing such large parity elsewhere and caching them will lead to both underutilization of bus and poor parity locality. Therefore, we adopt the design option to allow natural co-location of ECC/parity, in our case RS ECC bytes, with the data and cache MACs.

**RCNVM with eADR.** eADR [6] feature helps to flush CPU caches during power failure. When it comes to the recovery process, there are some differences between processor caches (L1, L2, and LLC) and metadata caches residing in memory controllers. Fortunately, the Metadata Cache can leverage several features, such as incremental counter update and inter-layer MAC relation, to facilitate recovery without relying on other costly methods. However, for CPU caches, the only possible options to ensure recovery are using frequent flush instructions, using extra power (eADR), or using a slow and costly non-volatile cache. In addition, the power budget for eADR needs to be deterministic and cannot rely on the assumption that not all parity cache blocks will be dirty during a failure. On the other hand, the L2 cache and LLC are more likely to absorb writes more frequently and can, therefore, maximize the power they are provided during a failure. This means that if we need to choose between protecting the parity cache and allocating an equal portion of the L2 cache for a given power budget, it is wiser to protect the L2 cache. It is important to note that the optimization that enables parity cache persistence without eADR is not practical for the L2 cache due to the higher number of L2 writes and the larger number of dirty blocks that change very frequently.

The question then arises as to whether optimizing hardware is better than spending power extravagantly, even though hardware optimization is possible. The current direction of modern processor design such as ARM v8 [29], M1 and M2 [27], Graviton [25], and other performance aware chips focuses on extracting maximum performance with minimal power consumption, not only in mobile processors but also in PC [67] and server processors [25]. This suggests that a one-time hardware optimization with negligible area overhead and a better Power-Performance pair is always the better choice.

**MMAC over data vs. encryption counters.** Another important question is why we cannot simply use MMAC over the counters instead of data MACs. In a typical situation, assuming we are operating within the ECC's correction limit, this would be possible. However, it is essential to note that RCNVM's motivation is to enable the recovery of MACs when using MAC-assisted reliability schemes, such as Synergy [45], which enhances performance and reliability by reusing the MAC as an error detector. Although this scheme is already proposed for DRAM systems, it is not feasible in NVM due to a recovery-time reliability bottleneck. If we use traditional methods of using MACs for security and ECC for errors instead, we do not need to recover the MACs. The cached MACs can be directly recalculated, assuming that BMT can authenticate the counter's integrity and ECC can guarantee the counter's error-correction. However, since we are using MACs for error detection, there will

be an unpredictable race hazard when trying to error-check and authenticate the counter before recalculating the MAC. Therefore, in RCNVM, we generate MMAC over data MACs so that the MACs can be used to verify and recover faulty data even in the event of a chip failure.

## 7 Related Works

The security aspects of NVMs have been studied in recent works, focusing on the efficient encryption methods [4], [50], crash consistency [31], [60], write optimization [64], and recovery time [10], [71]. Memory security in general, including counter mode encryption [56] and integrity protection [44], is also studied. Prior works related to NVM security assumed that the data MACs would be written to memory simultaneously when the data is written. Synergy [45] interchanges MAC and ECC positions in memory, storing MACs in the ECC chip while separately storing parities. Synergy also utilized the error detection capability of MACs to strengthen the reliability of the system.

Synergy does not reduce the number of writes as additional writes are now done for writing parities. The number of reads is reduced since errors are relatively rare, and parities are not read unless MAC detects any tampering or error in the data. Synergy does not use any write-back cache, which triggers more writes to the memory and is mostly designed for the DRAM system. Several systems, including cc-NVM [58], Osiris [60], and Anubis [71] are related to our work and propose crash consistency and recovery time reduction. However, each of these works is motivated by a different aspect. Osiris focuses on encryption counters recovery, and Anubis proposes low recovery time by introducing shadow tracking for security metadata. None of the works discussed the recovery of MACs and overlooked the MAC problem if cached in a write-back cache. Several research works are also done for the reliability exploration of emerging NVMs. For example, Zhang et el. [66] explored how chipkill reliability can be achieved in NVMs. Our work is the first to allow authenticated recovery of MACs, and it's the first to combine MACs into the integrity tree.

## 8 Conclusion

We present the ineffectiveness and recovery problem in the current implementation of MACs in secure NVM architectures. While the current assumptions incur either high overhead, cause design complexities, or can not achieve reliable recovery, our proposed scheme can achieve better performance and reliability with minor modifications in the memory controller. On average, RC-NVM can improve the performance by 18.2% over the state-of-the-art Synergy and reduce writes by 27%. Furthermore, RC-NVM is the first scheme to propose MAC recovery that can tolerate an entire chip failure during recovery and manages to recover all types of security metadata within seconds at the expense of marginal modifications within the memory controller.

## 9 Acknowledgments

Part of this work was funded through Office of Naval Research (ONR) grants N00014-21-1-2809 and N00014-21-1-2811, and the National Science Foundation (NSF) grants CNS-1814417, CNS-1908471 and CNS-2008339. 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] "Deprecating the PCOMMIT Instruction," https://software.intel.com/en-us/blogs/2016/09/12/deprecate-pcommit-instruction, accessed: 2019-11-13.

[2] "Introducing Micron® DDR5 SDRAM: More Than a Generational Update," https://www.micron.com/\protect\discretionary{\char\hyphenchar\font}{}{}/media/client/global/documents/products/white-paper/ddr5_more_than_a_generational_update_wp.pdf?la=en, accessed: 02/06/2021.

[3] "SPEC CPU 2006 Benchmarks," https://www.spec.org/cpu2006/, accessed: 2019-11-13.

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

[5] M. Al-Wadi, K. A. Zubair, D. Mohaisen, and A. Awad, "Phoenix: Towards ultra-low overhead, recoverable, and persistently secure NVM," *IEEE Trans. Dependable Secur. Comput.*, vol. 19, no. 2, pp. 1049–1063, 2022. [Online]. Available: https://doi.org/10.1109/TDSC.2020.3020085

[6] M. Alshboul, P. Ramrakhyani, W. Wang, J. Tuck, and Y. Solihin, "Bbb: Simplifying persistent programming using battery-backed buffers," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 111–124.

[7] A. Alsuwaiyan and K. Mohanram, "Mfnw: An mlc/tlc flip-n-write architecture," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 14, no. 2, pp. 1–32, 2018.

[8] M. Alwadi, A. Mohaisen, and A. Awad, "Promt: optimizing integrity tree updates for write-intensive pages in secure nvms," in *Proceedings of the ACM International Conference on Supercomputing*, 2021, pp. 479–490.

[9] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne, "Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers," *ACM SIGPLAN Notices*, vol. 51, no. 4, pp. 263–276, 2016.

[10] A. Awad, M. Ye, Y. Solihin, L. Njilla, and K. A. Zubair, "Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories," in *Proceedings of the 46th International Symposium on Computer Architecture, pages=104–115*, 2019.

[11] S. A. Berke, V. Sankaranarayanan, and B. M. Mutnury, "Maintaining highest performance of ddr5 channel with marginal signal integrity," Jun. 16 2020, uS Patent 10,685,736.

[12] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[13] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, and D. A. Patterson, "Raid: High-performance, reliable secondary storage," *ACM Computing Surveys (CSUR)*, vol. 26, no. 2, pp. 145–185, 1994.

[14] V. Costan and S. Devadas, "Intel SGX Explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[15] A. Freij, S. Yuan, H. Zhou, and Y. Solihin, "Persist level parallelism: Streamlining integrity tree updates for secure persistent memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 14–27.

[16] A. Freij, S. Yuan, H. Zhou, and Y. Solihin, "Persist level parallelism: Streamlining integrity tree updates for secure persistent memory," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 14–27.

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

[18] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, no. 5, pp. 91–98, 2009.

[19] M. N. Hsieh, A. F. Rodrigues, S. Li, K. Chen, N. Muralimanohar, C. D. Kersey, J. B. Brockman, and N. P. Jouppi, "System implications of memory reliability in exascale computing." Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), Tech. Rep., 2011.

[20] J. Huang and Y. Hua, "A write-friendly and fast-recovery scheme for security metadata in non-volatile memories," in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2021, pp. 359–370.

[21] R. Huang and G. E. Suh, "Ivec: off-chip memory integrity protection for both security and reliability," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 395–406, 2010.

[22] Intel, "Intel Optane DC Persistent Memory," "https://www.intel.com/content/www/us/en/architecture-and-technology/optane-dc-persistent-memory.html", 2020, [Online; accessed 05-February-2020].

[23] N. S. Islam, M. Wasi-ur Rahman, X. Lu, and D. K. Panda, "High performance design for hdfs with byte-addressability of nvm and rdma," in *Proceedings of the 2016 International Conference on Supercomputing*, 2016, pp. 1–14.

[24] J. Izraelevitz, J. Yang, L. Zhang, J. Kim, X. Liu, A. Memaripour, Y. J. Soh, Z. Wang, Y. Xu, S. R. Dulloor, J. Zhao, and S. Swanson, "Basic performance measurements of the intel optane DC persistent memory module," *arXiv preprint arXiv:1903.05714*, 2019.

[25] Q. Jiang, Y. C. Lee, and A. Y. Zomaya, "The power of arm64 in public clouds," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*. IEEE, 2020, pp. 459–468.

[26] R. Kateja, N. Beckmann, and G. R. Ganger, "Tvarak: software-managed hardware offload for redundancy in direct-access nvm storage," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 624–637.

[27] C. Kenyon and C. Capano, "Apple silicon performance in scientific computing," in *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2022, pp. 1–10.

[28] N. S. Kim, C. Song, W. Y. Cho, J. Huang, and M. Jung, "Ll-pcm: Low-latency phase change memory architecture," in *Proceedings of the 56th Annual Design Automation Conference 2019*, 2019, pp. 1–6.

[29] M. A. Laurenzano, A. Tiwari, A. Cauble-Chantrenne, A. Jundt, W. A. Ward, R. Campbell, and L. Carrington, "Characterization and bottleneck analysis of a 64-bit armv8 platform," in *2016 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2016, pp. 36–45.

[30] S. Lin and D. J. Costello, *Error control coding*. Prentice hall New York, 2001, vol. 2, no. 4.

[31] S. Liu, A. Kolli, J. Ren, and S. Khan, "Crash consistency in encrypted non-volatile main memory systems," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018.

[32] S. Liu, K. Seemakhupt, G. Pekhimenko, A. Kolli, and S. Khan, "Janus: Optimizing memory and storage support for non-volatile memory systems," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2019, pp. 143–156.

[33] X. Luo, D. Liu, K. Zhong, D. Zhang, Y. Lin, J. Dai, and W. Liu, "Enhancing lifetime of nvm-based main memory with bit shifting and flipping," in *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications*. IEEE, 2014, pp. 1–7.

[34] D. McGrew and J. Viega, "The galois/counter mode of operation (gcm)," *submission to NIST Modes of Operation Process*, vol. 20, pp. 0278–0070, 2004.

[35] T. Mittelholzer, M. Stanisavljevic, N. Papandreou, and H. Pozidis, "High-throughput ecc with integrated chipkill protection for non-volatile memory arrays," in *2021 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[36] S. Nalli, S. Haria, M. D. Hill, M. M. Swift, H. Volos, and K. Keeton, "An analysis of persistent memory use with whisper," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 135–148, 2017.

[37] P. M. Palangappa and K. Mohanram, "Flip-mirror-rotate: An architecture for bit-write reduction and wear leveling in non-volatile memories," in *Proceedings of the 25th edition on Great Lakes Symposium on VLSI*, 2015, pp. 221–224.

[38] V. Prabhakaran, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Analysis and evolution of journaling file systems." in *USENIX Annual Technical Conference, General Track*, vol. 194, 2005, pp. 196–215.

[39] M. K. Qureshi, S. Gurumurthi, and B. Rajendran, "Phase change memory: From devices to systems," *Synthesis Lectures on Computer Architecture*, vol. 6, no. 4, pp. 1–134, 2011.

[40] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali, "Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling," in *2009 42nd Annual IEEE/ACM international symposium on microarchitecture (MICRO)*. IEEE, 2009, pp. 14–23.

[41] J. Rakshit and K. Mohanram, "ASSURE: Authentication scheme for secure energy efficient non-volatile memories," in *Proceedings of the 54th Annual Design Automation Conference 2017*, 2017, pp. 1–6.

[42] P. Rogaway and J. Black, "Pmac: A parallelizable message authentication code," *Preliminary Draft, October*, vol. 16, 2000.

[43] P. Rogaway and J. Black, "Proposal to nist for a parallelizable message authentication code," 2001.

[44] B. Rogers, S. Chhabra, M. Prvulovic, and Y. Solihin, "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)*. IEEE, 2007, pp. 183–196.

[45] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, and M. K. Qureshi, "Synergy: Rethinking secure-memory design for error-correcting memories," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 454–465.

[46] S. Schechter, G. H. Loh, K. Strauss, and D. Burger, "Use ecp, not ecc, for hard failures in resistive memories," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 141–152, 2010.

[47] H. Seol, M. Kim, Y. Kim, T. Kim, and L.-S. Kim, "Amnesiac dram: A proactive defense mechanism against cold boot attacks," *IEEE Transactions on Computers*, 2019.

[48] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi, "Memory errors in modern systems: The good, the bad, and the ugly," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 297–310, 2015.

[49] V. Sridharan and D. Liberty, "A study of dram failures in the field," in *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. IEEE, 2012, pp. 1–11.

[50] S. Swami and K. Mohanram, "ACME: Advanced counter mode encryption for secure non-volatile memories," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE, 2018, pp. 1–6.

[51] S. Swami and K. Mohanram, "ARSENAL: Architecture for Secure Non-Volatile Memories," *IEEE Computer Architecture Letters*, vol. 17, no. 2, pp. 192–196, 2018.

[52] M. Taassori, R. Balasubramonian, S. Chhabra, A. R. Alameldeen, M. Peddireddy, R. Agarwal, and R. Stutsman, "Compact leakage-free support for integrity and reliability," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 735–748.

[53] Z. Wang, C.-H. Choo, M. A. Kozuch, T. C. Mowry, G. Pekhimenko, V. Seshadri, and D. Skarlatos, "Nvoverlay: Enabling efficient and scalable high-frequency snapshotting to nvm."

[54] J. Xu, L. Zhang, A. Memaripour, A. Gangadharaiah, A. Borase, T. B. Da Silva, S. Swanson, and A. Rudoff, "Nova-fortis: A fault-tolerant non-volatile main memory file system," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 478–496.

[55] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang, and H. Li, "Emerging non-volatile memories: Opportunities and challenges," in *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 2011, pp. 325–334.

[56] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2. IEEE Computer Society, 2006, pp. 179–190.

[57] F. Yang, Y. Chen, H. Mao, Y. Lu, and J. Shu, "Shieldnvm: An efficient and fast recoverable system for secure non-volatile memory," *ACM Transactions on Storage (TOS)*, vol. 16, no. 2, pp. 1–31, 2020.

[58] F. Yang, Y. Lu, Y. Chen, H. Mao, and J. Shu, "No Compromises: Secure NVM with Crash Consistency, Write-Efficiency and High-Performance," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2019, pp. 1–6.

[59] J. Yao, H. Jiang, Q. Cao, L. Tian, and C. Xie, "Elastic-raid: A new architecture for improved availability of parity-based raids

by elastic mirroring," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 1044–1056, 2015.

[60] M. Ye, C. Hughes, and A. Awad, "Osiris: A low-cost mechanism to enable restoration of secure non-volatile memories," in *51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2018)*, 2018.

[61] M. Ye, K. A. Zubair, A. Mohaisen, and A. Awad, "Towards low-cost mechanisms to enable restoration of encrypted non-volatile memories," *IEEE Trans. Dependable Secur. Comput.*, vol. 18, no. 4, pp. 1850–1867, 2021.

[62] S. F. Yitbarek, M. T. Aga, R. Das, and T. Austin, "Cold boot attacks are still hot: Security analysis of memory scramblers in modern processors," in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2017, pp. 313–324.

[63] D. H. Yoon, N. Muralimanohar, J. Chang, P. Ranganathan, N. P. Jouppi, and M. Erez, "Free-p: Protecting non-volatile memory against both hard and soft errors," in *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 2011, pp. 466–477.

[64] V. Young, P. J. Nair, and M. K. Qureshi, "DEUCE: Write-efficient encryption for non-volatile memories," *ACM SIGPLAN Notices*, vol. 50, no. 4, pp. 33–44, 2015.

[65] J. Yue and Y. Zhu, "Accelerating write by exploiting pcm asymmetries," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2013, pp. 282–293.

[66] D. Zhang, V. Sridharan, and X. Jian, "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, 2018, pp. 710–723.

[67] Z. Zhang, "Analysis of the advantages of the m1 cpu and its impact on the future development of apple," in *2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*. IEEE, 2021, pp. 732–735.

[68] M. Zheng, J. Tucek, F. Qin, and M. Lillibridge, "Understanding the robustness of ssds under power fault," in *11th {USENIX} Conference on File and Storage Technologies ({FAST} 13)*, 2013, pp. 271–284.

[69] J. Zhou, A. Awad, and J. Wang, "Lelantus: fine-granularity copy-on-write operations for secure non-volatile memories," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 597–609.

[70] Y. ZOU, K. A. ZUBAIR, M. ALWADI, R. M. SHADAB, S. GANDHAM, A. AWAD, and M. LIN, "Ares: Persistently secure non-volatile memory with processor-transparent and hardware-friendly integrity verification and metadata recovery," *ACM Transactions on Embedded Computing Systems*, vol. 1, no. 1, 2021.

[71] K. A. Zubair and A. Awad, "Anubis: ultra-low overhead and recovery time for secure non-volatile memories," in *Proceedings of the 46th International Symposium on Computer Architecture*. ACM, 2019, pp. 157–168.

[72] K. A. Zubair, S. Gurumurthi, V. Sridharan, and A. Awad, "Soteria: Towards resilient integrity-protected and encrypted non-volatile memories," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 1214–1226.

[73] K. A. Zubair, D. Mohaisen, and A. Awad, "Filesystem encryption or direct-access for NVM filesystems? let's have both!" in *IEEE International Symposium on High-Performance Computer Architecture, HPCA 2022, Seoul, South Korea, April 2-6, 2022*. IEEE, 2022, pp. 490–502.

[74] P. Zuo, Y. Hua, and Y. Xie, "SuperMem: Enabling application-transparent secure persistent memory with low overheads," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 479–492.

**Kazi Abu Zubair** obtained his PhD in Computer Engineering from NC State University in 2022, where he was advised by Prof. Amro Awad within the Secure and Advanced Computer Architecture (SACA) research group. He is currently a CPU Core & Simulator Architect at Intel Corporation. is a final year PhD student majoring in Computer Engineering at NC State. His research interests include secure memory architecture, NVM security, and memory reliability. He received his BS degree from the University of Chittagong, Bangladesh, and worked in the R&D industry for several startup companies in Bangladesh before joining SACA.

**Rahaf Abdullah** is a PhD student in computer engineering at NC State University. Her research interests revolve around memory security in addition to the security of GPUs with the least overheads possible. Rahaf graduated with a BS degree in Computer Engineering from An-Najah National University in Nablus, Palestine, at the end of 2020. Prior to joining NC State University, she worked as a software engineer developing macOS applications used in the production of laptops and smart devices.

**David Mohaisen** obtained his Ph.D. in Computer Science from the University of Minnesota in 2012. He is currently a professor of Computer Science at the University of Central Florida, where he leads the Security and Analytics Lab (SEAL) and has been since 2017. Previously, he was an Assistant Professor at SUNY Buffalo (2015-2017) and a Senior Scientist at Verisign Labs (2012-2015). His research interests are in the broad area of applied security and privacy, covering aspects of computer and networked systems, software systems, IoT and AR/VR, and machine learning. His research has been supported by NSF, NRF, AFRL, AFOSR, etc., and has been published in top conferences and journals alike, with multiple best paper awards. His work was featured in the New Scientist, MIT Technology Review, ACM Tech News, Science Daily, etc. Among other services, he is currently an Associate Editor of IEEE Transactions on Mobile Computing and IEEE Transactions on Parallel and Distributed Systems. He is a senior member of ACM (2018) and IEEE (2015), a Distinguished Speaker of the ACM and Distinguished Visitor of the IEEE Computer Society.

**Amro Awad** is currently an assistant professor and leads the Secure and Advanced Computer Architecture (SACA) research group at NC State. Before joining NC State in Fall 2020, he was an assistant professor at UCF (2017-2020). Amro received his PhD degree from NC State University in 2016. During his Ph.D., he had several stints at LANL, HP Labs and AMD Research. His research papers have been published in top-venues in computer architecture, such as ISCA, MICRO, HPCA, ASPLOS, PACT and ICS. His research group has been funded by NSF, NSWCDD, ONR, DARPA, Sandia National Laboratories, US Army CCDC, and AFRL. He pioneered the area of persistently-secure memories, where the security, high-availability and crash consistency need to be guaranteed in future memory systems. He also was the PI of DARPA's MemSec project resulting in dozens of novel ideas.