Minerva: Rethinking Secure Architectures for the Era of Fabric-Attached Memory Architectures

Mazen Alwadi[1], Rujia Wang[2], David Mohaisen[3], Clayton Hughes[4], Simon David Hammond[4], Amro Awad[5]
Jordan University of Science and Technology[1], Illinois Institute of Technology[2], University of Central Florida[3],
Sandia National Laboratories[4], North Carolina State University[5]
mgalwadi@just.edu.jo, rwang67@iit.edu, mohaisen@ucf.edu, {chughes,sdhammo}@sandia.gov, ajawad@ncsu.edu

2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 978-1-6654-8106-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPDPS53621.2022.00033

***Abstract*—Fabric-attached memory (FAM) is proposed to enable the seamless integration of directly accessible memory modules attached to the shared system fabric, which will provide future systems with flexible memory integration options, mitigate underutilization, and facilitate data sharing. Recently proposed interconnects, such as Gen-Z and Compute Express Link (CXL), define security, correctness, and performance requirements of fabric-attached devices, including memory. These initiatives are supported by most major system and processor vendors, bringing widespread adoption of FAM-enabled systems one step closer to reality and security concerns to the forefront.**

This paper discusses the challenges for adapting secure memory implementations to FAM-enabled systems for the first time in literature. Specifically, we observe that handling the security metadata used to protect fabric-attached memories needs to be done deliberately to eliminate unintentional integrity check failures and/or security vulnerabilities, caused by an inconsistent view of the shared security metadata across nodes. Our scheme, Minerva, elegantly adapts secure memory implementations to support FAM-enabled systems with negligible performance overheads (3.8% of an ideal scheme), compared to the performance overhead (99.5% of an ideal scheme) for a scheme that uses conventional invalidation-based cache coherence to ensure the consistency of security metadata across nodes.

I. Introduction

The continued growth of large-scale scientific computing, diversity of workloads, and virtual machine consolidation all points to an increased need for larger memory capacity [28]. To increase the accessible memory for each computing node, minimize the memory under-utilization, and reduce the memory access latencies researchers proposed the disaggregation of memory modules from the computing nodes [23]. The increasing popularity of memory disaggregation led major vendors to formalize a connection fabric i.e., Gen-Z [6], CCIX [2], CXL [3]. Such fabrics enable a new system architecture where computing nodes and memory modules are connected to the system's fabric, in which each processing node can access the memory modules without passing through other processing nodes. The huge memory capacity requirement for Fabric-Attached Memory (FAM) architectures, makes the emerging Non-Volatile Memories (NVMs) an appealing option as one of the main building blocks of the main memory.

The emerging NVMs stand as an appealing candidate for several reasons. Such NVMs have access latencies comparable to DRAM and higher densities, where each DIMM is expected to be in the order of hundreds of gigabytes or even terabytes i.e., each Optane DC module can have 512GB of capacity [9]. Additionally, these NVMs are byte-addressable, have lower cost per bit, ultra-low idle power consumption, and retain data during power loss episodes [14], [25]–[27], however; at a limited write endurance. Additionally, these NVMs have slow and power-consuming writes [12], [14]–[16], [40], [41]. While the NVM's data persistency might be the most appealing feature, it also facilitates data remanence attacks. As such, NVMs are typically coupled with security features to ensure the integrity and confidentiality of the data [12], [14], [40], [41].

Secure memory implementations limit the trust boundaries to the processor chip only, which requires ensuring data confidentiality and integrity when it leaves the processor chip [14], [40]. Such security features are gaining more popularity in the research community [12], [16], [29], [35], [36], [40], [41] and in industrial products e.g., Intel's Software Guard Extension (SGX) [8], [18], Intel's Total Memory Encryption (TME) [7], and AMD's Secure Memory Encryption (SME) [1]. Such increasing popularity of these security features is mainly caused by the increased attack surface in current computer systems, the proliferation of accelerators, current computer systems comprising hundreds of sub-systems that could be manufactured by different vendors [1], [13]. Additionally, the data remanence attacks which were performed against the DRAM in the form of cold-boot attack [21], and are easier to implement against NVMs, are making the security features a functional requirement of the emerging NVMs [14], [15], [40], [41]. Secure memory implementation adopts an encryption scheme to protect the data confidentiality and an integrity tree to ensure its integrity [35], [37], [40], [41]. However, these security features entail high performance overheads, a large number of memory writes, and have crash consistency issues when used with NVMs [12], [14], [29], [35], [37], [41].

Reducing the overheads of secure memory implementations has been driving the efforts of researchers for the past decade. However, the proposed schemes to reduce the performance overheads and increase the cachability of integrity tree nodes and encryption counters [14], [18], [33], [35]–[37], recover the encryption counters [29], [40], and recover the integrity tree [12], [41] all assumed the memory is accessed *directly* by a single Processing Element (PE). With FAM architectures

This work was done when Mazen was working under the supervision of Amro Awad at UCF. Amro Awad is now with the ECE Department at NC State.

1530-2075/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPS53621.2022.00033

deviating from this assumption by allowing multiple PEs to access the memory directly, these schemes challenge one of the most fundamental assumptions in secure memory implementation. Such deviation results in a security metadata coherence problem, which can lead to unorthodox overheads for several reasons. First, unlike data coherence, the security metadata is transparent to software, which mandates a hardware solution. Second, for each data cacheline write, its associated encryption counter and the whole branch of the integrity tree need to be updated, which can exceed 12 levels for such huge memories. Third, typical data coherence can be solved by notifying only the sharers of the modified cacheline. On the other hand, the security metadata is shared by all the PEs sharing the memory. Thus, for each cacheline update, the associated security metadata (12 cachelines) needs to be propagated to all the PEs in the system. Fourth, failing to communicate these updates in a timely manner can compromise the system's security by allowing encryption counter reuse, replay attacks, and detecting legitimate memory updates as tampering and locking the whole system. Finally, to ensure the system's ability to recover from crashes, in case of NVMs, these updates need to be persisted to the NVM, which can exacerbate the NVM's write endurance problem and significantly reduce its expected lifetime.

To address this problem, we propose a novel hardware support, Minerva, which enables both coherent and crash-consistent views of the security metadata in FAM architectures. In particular, Minerva exploits the applications' data locality and the integrity tree nodes' coverage to minimize the overhead of maintaining security metadata coherence. Furthermore, Minerva ensures security at the system level by enforcing exclusive caching of security metadata. In other words, Minerva allows a single security metadata cacheline to be cached by one processing element at any point in time. To reduce the overhead of maintaining coherence, Minerva relies on a lazy-invalidate scheme to make the overhead a function of security metadata cache *misses* instead of security metadata cache *updates*. Finally, to reduce the communication overhead and facilitate scalability, Minerva implements an inexpensive directory-like scheme that leverages the unused bits in the parallelizable integrity tree to store the ID of the PE currently caching the node.

To evaluate Minerva, we use a publicly available architectural simulator, the Structural Simulation Toolkit (SST) [32], which models disaggregated memory systems and is widely used in relevant architectural studies [24]. We model a FAM system with multiple PEs, and, similar to previous work [30], we use 7 in-house developed persistent applications that stress the memory system. On average, Minerva's performance is 3.8% slower than a hypothetical baseline that ensures consistency of security metadata across PEs without any changes to the existing implementations of secure memory. Additionally, we used applications from the SPEC2006 benchmark suite and HPC workloads. Meanwhile, the performance of a scheme that leverages conventional invalidation-based cache coherence incurs an average of 99.5% performance overhead. Moreover, Minerva supports crash consistency for FAM architectures that can leverage persistent memories. In summary, the contributions of our work are as follows:

- We propose Minerva, a novel hardware support for security metadata coherence in FAM architectures, which makes the security metadata coherence messages a function of security metadata cache *misses* instead of security metadata cache *updates*, and enables decentralized updates of the security metadata.
- We design a distributed security metadata cache recovery scheme, enabling Minerva's lazy-invalidate scheme, crash consistency, and system recovery.
- We evaluate other possible schemes that can provide the same security and crash consistency requirements and compare them against Minerva.
- We analyze Minerva's impact on the system performance and vary the number of processing elements in the system to provide insight on Minerva's scalability.

II. Background and Motivation

In this section, we discuss the most related concepts followed by the motivation of our work.

A. Background

1) Threat Model: We assume an attacker capable of passive and active attacks. Similar to state-of-the-art work in secure memory architecture [35]–[37], [40], [41], the attacker is capable of scanning the memory to read its content, snoop the memory bus, drop packets, tamper with the packets or memory content, and replay old packets. Finally, memory access pattern leakage [39], timing side-channel leakage [39], power analysis [31], and malicious PE(s) are beyond the scope of the paper. However, our proposed secure architecture scheme is orthogonal to defenses targeting these types of attacks, thus it can be integrated along with defenses against these attacks to achieve a higher degree of protection.

2) Connecting Fabric Protocols: To address the increasing demand for higher memory bandwidth, larger memory capacities, and memory sharing requirements, Gen-Z was developed [6]. Gen-Z is a new data access architecture that was developed by a growing consortium of technological companies who believe memory-semantic data access is crucial to address the growing needs [4]. The Gen-Z enables all types of components to communicate with each other directly using a unified protocol, which eliminates the need for intermediate interconnects, simplifies the communication, and improves the interoperability [4]. In order to utilize the Gen-Z fabric, each component needs to implement the Gen-Z protocol interface at the memory controller, which translates high-level commands i.e., memory read/write operations into the fabric commands. Additionally, the Gen-Z protocol has several security features, which include packets encryption and end-to-end integrity verification [5]. Furthermore, Gen-Z breaks the processor-memory interlock, which abstracts the memory media and enables the disaggregation of memory modules [4].

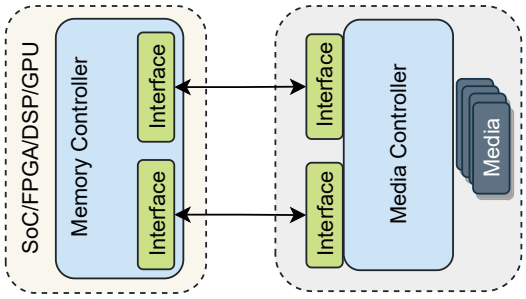

Fig. 1. Gen-Z memory controller.

In the split-controller design, shown in Figure 1, the memory controller still performs almost all the tasks in current processors. However, the media controller is responsible for media-specific operations, executing the commands, and returning responses [4]. Additionally, the Gen-Z protocol can be used with multiple topologies and deployment models i.e., P2P, Daisy chain, Mesh, Torus, switched or any routing topology, which enables the memory-centric architectures [4].

3) Memory-Centric Architectures: Current High-Performance Computing (HPC) systems are implemented such that each computing node has its own memory module(s) attached to it directly. One of the main issues with this architecture is that memory modules are only utilized by the processor attached to it, as accessing data in other nodes is typically implemented through expensive network interfaces [24]. This approach of coupling memory modules to processors limits the efficiency, flexibility, and performance of current systems. Therefore, system vendors are moving toward memory-centric architectures [2], [6], [17], [19] i.e., HPE Labs' The Machine [22], Facebook's Disaggregated Rack [38], and Intel's Rack Scale Architecture [28].

Memory-centric architectures, also known as Fabric-Attached Memory (FAM) architectures, allow multiple processing elements to connect to a shared memory pool using a connecting fabric, and seamless integration of processing elements from different vendors. The main enabler of the FAM architectures is a connecting fabric, which is governed by a connecting protocol, such as Gen-Z [6], Compute Express Lanes (CXL) [3] or Cache Coherent Interconnect for Accelerators (CCIX) [2].

In FAM architectures, each component can communicate with any other component directly by implementing the Gen-Z interface. As FAM architectures are expected to have large shared memory pools and due to the high power requirements for frequent refresh and cooling of DRAM, FAM architectures are expected to use emerging NVMs as the foundation of the shared memory or as a part of it [22]. While using secure memory measures is not limited to the NVM, but due to the NVM's data persistency, which facilitates data remanence attacks, the use of memory encryption and integrity verification becomes a necessity.

4) Secure Memory Architectures: Secure memory architecture limits the trust boundary to the processor chip, therefore the memory content is usually encrypted to ensure data confidentiality, and a Merkle tree (MT) is used to verify the data's integrity. Below we describe the state-of-the-art schemes in memory encryption (counter-mode encryption), and integrity verification (Merkle tree).

Counter-mode encryption is used in state-of-the-art processor systems as it provides strong defenses against a wide range of attacks. Namely, the counter-mode encryption thwarts the dictionary-based attacks, known plain-text attacks, and snooping attacks [14], [40], [41]. Moreover, the counter-mode encryption reduces the decryption latency by overlapping the decryption with the memory read latency. To ensure the security of the counter-mode encryption, reusing encryption counters (ECs) is prohibited, as it facilitates known plaintext attacks [14], [40].

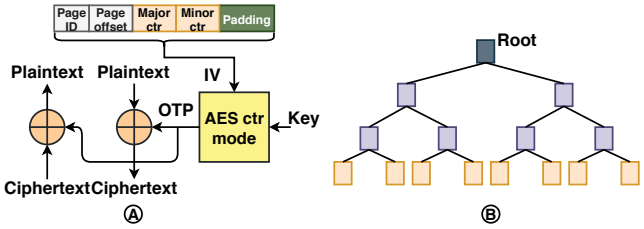

Fig. 2. (a) Split counter-mode encryption, (b) Bonsai Merkle tree (BMT).

Figure 2:(A) shows the split-counter mode encryption and how it works. In the split counter mode, an initialization vector (IV) composed of a per-page major counter, per-block (cacheline) minor counter, page offset, page ID, and padding is encrypted by an AES encryption engine using an encryption key to generate a One-Time-Pad (OTP). The OTP is then XORed with the plaintext/ciphertext to complete the encryption/decryption [18], [29], [33], [36].

Merkle trees (MT) are used in state-of-the-art schemes [14], [29], [37], [40], [41] for memory integrity verification purposes. Depending on the tree structure, Merkle trees can be non-parallelizable (e.g., General Merkle tree) or parallelizable (e.g., SGX style counters tree). Additionally, the MT can be built on top of the encryption counters instead of the data, to reduce the MT size. Wherein the integrity of the encryption counters is protected by the MT, and the integrity of the data is protected using keyed-Message Authentication codes (HMACs). This organization of the MT is typically referred to as the Bonsai-Merkle Tree (BMT) [33].

Merkle tree is an N-ary hash tree where the hashes of the data blocks and/or encryption counters are the leaves and every N leaves will have a hash value calculated to generate the next tree level. Similarly, all the intermediate nodes up to the root are constructed using the hash value based on its children. The root is always kept secure, that is, it never leaves the chip. Any tamper with the data used to generate the Merkle tree root, data blocks and/or encryption counters, leads to the failure of regenerating the same root value.

In general, the hashes of each level of the Merkle tree are calculated using the hashes of the level below, which requires a

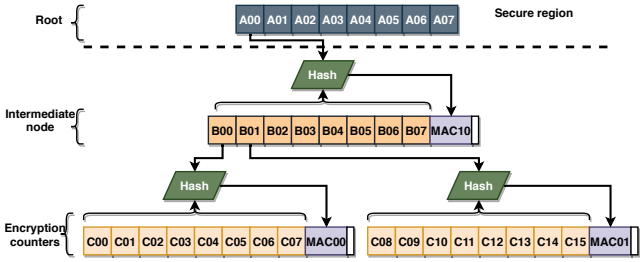

Fig. 3. Tree of counters.

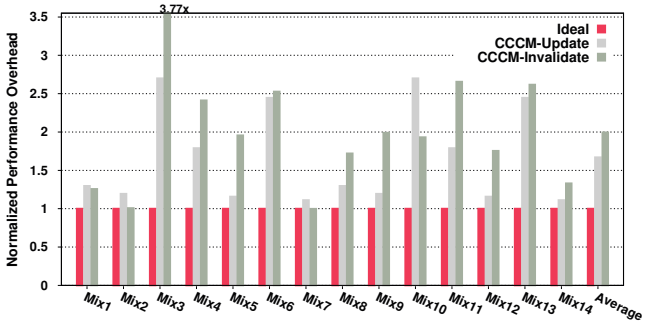

Fig. 4. The performance overhead of traditional coherence schemes.

sequential update of the tree. Figure 2:(B) shows a 2-ary BMT. As the figure shows, the bottom level includes the encryption counters which are hashed to generate the first level of the tree, then each two hashes from the first level are hashed to generate the second level. The process then continues recursively until a single node is generated, the root.

On the other hand, the parallelizable tree (known as Tree-of-Counters (ToC)) works differently; ToC includes eight encryption counters and a Message Authentication Code (MAC) value in each leaf node. To generate the associated MAC value, the node counters, and a version from the parent level are hashed together. Figure 3 shows a subset of the ToC structure. Notice that, in ToC, whenever a data block is written back to the memory, its associated counter is incremented and the parent, also known as the version of the counter, is incremented as well [12], [18], [20], [41].

B. Motivation

Maintaining a coherent view of the security metadata in a FAM architecture is very challenging for several reasons. First, the use of software data coherence solutions is inapplicable due to the transparency of security metadata to software. Second, for each cacheline update, its associated encryption counter and the whole MT branch need to be updated. Note that the MT size grows as the memory size grows, which can reach 12 levels for a 1TB memory. Assuming a system with P PEs where the memory is protected by n levels MT, whenever a data cacheline is updated, $(n-1)*P$ coherence messages need to be sent to notify the PEs about the update. Third, to ensure the correctness of the MT updates and prevent data races, the whole MT branch needs to be locked before updating the MT, and the updates need to be held in the processor write buffers until the update is acknowledged by all the PEs in the system, which can lead to filling the processor buffers and may cause processor stalls due to the inability to replace security metadata cachelines. Fourth, while the PEs are sharing the memory, they do not necessarily need to share the data, which means not all the MT branch updates need to be reflected. However, the MT node's coverage increases based on its level. For instance, assuming P PEs sharing a 1TB memory that is protected by a 12-levels MT, two different PEs with interleaved memory pages will be sharing 10-11 levels of the MT. On the other hand, two different PEs with each one operating on adjacent 1GB of the shared memory would still need to share the top 4 levels of the MT. In the best case, where they are operating on separate MT branches, the PEs would still have to share the root, thus communicating the root updates would still be necessary. Finally, for crash consistency requirements for NVMs, these updates need to be persisted atomically along with the modified data cacheline. As failing to persist part of the integrity tree branch, the encryption counter, or the data cacheline can fail the integrity verification, which can lead to either discarding the memory content or accepting the risk that the memory content might be tampered with.

In order to evaluate the performance overhead for achieving security metadata coherence, we designed two Crash Consistent Coherence Messages (CCCM) schemes; namely, CCCM-Invalidate and CCCM-Update. The CCCM-Invalidate scheme is based on the MESI protocol, which sends invalidation coherence messages to other PEs in the system whenever a cacheline is updated. The updated cachelines are held in the processor write buffer until the invalidation is acknowledged by all other PEs in the system. The CCCM-Update scheme works in a similar manner, but uses update messages instead of the invalidation. Both schemes ensure the system's ability to recover by persisting the writes to the NVM in an atomic manner. As shown in Figure 4, using traditional coherence schemes can have 2x slowdown.

III. Design

In this section, we discuss Minerva's design in light of the threat model discussed in II-A1, possible design options, and their trade-offs. Before discussing Minerva's design, we start with the security requirements followed by possible design options.

A. Security Requirements

Secure memory implementation limits the trust base to the processor chip only, which requires protecting the data integrity and confidentiality when the data leave the processor chip. In FAM architecture, having multiple PEs in the system introduces several attacks that are not possible in a processor-centric architecture. To protect against these attacks, the design should meet the following requirements:

- **Ensure the Data Integrity:** as each PE is caching a subset of the security metadata, the system should ensure

all PEs are able to verify the data integrity using the most recent version of the MT nodes. Thus, the system should allow PEs to communicate without PE-PE packets dropping.

- **Prevent Encryption Counter Reuse:** as encryption counter reuse compromises the counter mode's security, which can happen if multiple PEs are updating the same EC simultaneously.
- **Ensure Correct Updates of the MT:** having multiple PEs updating the MT simultaneously can lead to incorrect updates of the MT branch if ordering was not followed strictly.

Note that these requirements are hard to achieve in FAM architectures due to having multiple PEs, wherein each PE is caching a subset of the security metadata, where they need to communicate in a timely and secure manner.

B. Design Requirements

The design should meet the design requirements necessary to allow wide adoption and high-performance while ensuring the system's security requirements. In summary, the design requirements are as follows:

- **High-Performance:** the design should allow secure memory implementation with minimal performance overhead.
- **Memory Utilization:** increasing the memory utilization requires the design to enable PEs to share the memory, even if the running applications are not sharing the same pages.
- **Scalability:** addressing the coherence problem should be done with a minimal number of exchanged messages to allow scalability.
- **Crash Consistency:** this requirement is concerned with ensuring a consistent state of the data and its associated security metadata to allow recoverability.
- **NVM Friendly:** this requirement is concerned with ensuring the previous requirements with minimal number of writes to avoid shortening the NVM's lifetime.

C. Strawman Design Options and Limitations

In the following, we discuss several straightforward design options which can potentially meet the design requirements.
Option 1: Centralized security metadata handling achieved by having one PE handling all the encryption and integrity verification operations. While such a scheme can eliminate the security metadata coherence problem, it can create a single point of failure at the trusted PE and increase the performance overhead due to throttling the trusted PE. Additionally, this technique defies the purpose of FAM, where PEs can access the shared memory directly. Similarly, the security metadata handling can be done using in/near memory computing techniques, while using a point-point encryption to protect the data in transient as proposed by Awad et al. [16]. However, such a scheme does not limit the trust base to the PEs only, but expands it to include the logic in the memory modules.

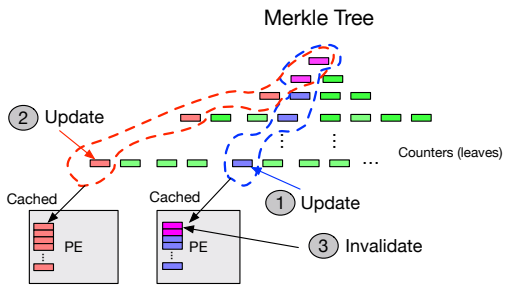

Fig. 5. Update process in invalidation-based security metadata coherence protocols.

Option 2: Allocate specific memory region per PE where each region is protected with different ECs and a separate MT. While this scheme will eliminate the security metadata coherence problem, such a scheme have several drawbacks. First, it prevents the PEs from effectively sharing the memory and reduces the memory utilization, which defies the purpose of FAM. Second, accessing a memory region that belongs to another PE requires going through the owner PE, or moving the data from the owner's memory region to the requester's memory region, which requires re-encrypting the data using the requester's metadata. Alternatively, the owner can share its encryption key with the requester to allow access to the owner's region, which would require coherence between the two. Third, such a scheme requires complex key management protocol to handle the credentials used by each PE.
Option 3: Utilize existing coherence mechanisms used for data coherence. While data coherence can be achieved using software or hardware techniques, security metadata is transparent to software which limits the possible solutions to hardware techniques only. Hardware coherence mechanisms can be classified into invalidate- or update-based schemes. While the invalidate-based schemes i.e., MESI/MOESI are more popular in data coherence, due to the reduced traffic, using such schemes can lead to higher overheads in case of security metadata, due to frequently invalidating shared metadata cachelines. Figure 5 illustrates the effect of having multiple PEs updating different data files in a shared memory region. Note that despite the PEs are updating different files, they still affect each other, due to sharing the same MT as discussed in section II-B.

While the first option meets the security requirements, is NVM friendly, crash consistent, and can provide good memory utilization, it lacks the required scalability and will have a high performance overhead. On the other hand, the second option can meet the security requirements, is NVM friendly, scalable, crash consistent, and have a low performance overhead, but it does not allow the memory sharing and thus reduces the memory utilization. Finally, the last design option supports memory utilization and crash consistency, but lacks the scalability, is not NVM friendly, and has a high performance overhead as shown in Figure 4. Additionally, this option requires using a universal system clock and messages

integrity protection to meet the security requirements.

D. Minerva's Design

Before delving into the details of Minerva, we discuss how Minerva meets the design requirements, and maintains the system's security against the previously mentioned attacks.

Maintain system security. Minerva uses counter-mode encryption to ensure the confidentiality, and implements a Tree of Counters (ToC) to ensure the integrity. While these measures can protect the data at rest, dropping PE-PE data packets can lead to ECs reuse and integrity verification failures. Fortunately, the Gen-Z fabric provides an end-end authentication and integrity verification, which ensures the packets delivery and integrity. Combining these techniques Minerva ensures the system's security, but has a high performance overhead.

Reduce performance overhead. The performance overhead of maintaining a security metadata coherence stems from ① the number of processing elements caching the updated cachelines, and ② the number of updated cachelines for each memory write. In order to reduce the number of PEs caching a security metadata cacheline, Minerva enforces exclusive caching, which means a security metadata cacheline can be cached by one PE only at any point of time. The exclusive caching allows each PE to operate on its cached security metadata without notifying other PEs for each update. However, exclusive caching does not reduce the number of updated security metadata cachelines for each memory write. Thus, Minerva uses a lazy-update scheme to update the ToC, which can limit the number of updated security metadata cachelines to one for each write.

Improve the scalability. Despite the advantages of the lazy-update scheme, updating a security metadata cacheline parent would still be required on eviction. Thus, whenever a PE requests a ToC node from another PE, the responder has to update the parent of the requested ToC node, which can lead to a chain of requests and updates. Minerva expands the view of security metadata into a system-level instead of PE-level, which allows the PEs to transfer ToC nodes to each other without the need to update the parent of the transferred node. Combining these techniques results in a system-level scheme, which we refer to as the lazy-invalidate scheme.

Crash consistency support. While the lazy-invalidate scheme can change the security metadata coherence messages to a function of the security metadata cache *misses* instead of *updates*, its still susceptible to crash consistency problems and requires frequent coherence broadcasts to request security metadata nodes that are cached in other PEs. Therefore, Minerva implements a distributed crash consistency scheme that incorporates a Phoenix-like [12] persist unit in each PE, as discussed in section III-D2, and implements a cheap directory scheme discussed in section III-D1.

1) Node Ownership Tracking: Minerva employs a directory-like scheme by maintaining the owner-ID of each cached security metadata node in the memory. Note that providing security measures for the region containing the owner-IDs will result in broadcasting the updates each time a node's owner-ID is updated. Therefore, Minerva uses a dedicated unprotected memory region for general MT. On the other hand, each ToC node contains an unused 8-bit space which can be used to tag the node with the current caching PE, therefore we will be focusing on the ToC, although the same idea applies to the general MT.

The unused 8 bits, shown in Figure 6, can be used to write the ID of the PE currently caching the node, or the memory ID if not cached by any PE. Thus, when a security metadata cache miss happens, the requesting PE reads the node's owner-ID tag from the memory, then sends a read-invalidate request to owner. Upon receiving the request, the owner writes the requester's ID as the node owner-ID, then sends the requested node. Note that the transfer operation is done as a transaction to prevent data races and using the owner-ID tag will reduce the communication whenever a PE requests a node cached at another PE. For nodes that are memory-tagged, the request has to be broadcasted to prevent duplicate node's caching.

As mentioned earlier, the owner-ID tag bits are not protected, therefore an attacker can tamper with these bits. However, tampering with these bits will result in requesting a node from a PE that does not have it in its cache, which leads to a broadcast. Therefore, tampering will be detected and it will only lead to a broadcast. On the other hand, protecting the owner-ID bits will cause more overhead than broadcasting. Even though tagging each node with the owner-ID can eliminate the broadcasts for nodes cached by other PEs, broadcasting the request of a memory owned node, and the root each time it is updated would still be required to prevent replay attacks. Note that Minerva's exclusive caching reduces the traffic and removes the need to notify other PEs when a security metadata cacheline is updated.

Note that handling the owner-ID updates can be done as described earlier, or by implementing a directory functionality in the memory modules. While using a passive memory and delegating the owner-ID updates to the PEs leads to data races when requesting the nodes from the owner PE, implementing a directory functionality in the memory modules can lead to another type of data races when a node's owner-ID is updated but before the PE actually obtains the node from its previous owner. However, both data races can be resolved by performing the owner-ID update as a transaction, and notifying other requesting PEs about the node being updated.

2) Distributed Crash Consistency and Recovery: Minerva has multiple PEs transferring MT nodes to each other upon request. Therefore, Minerva employs a persist unit similar to Phoenix [12] for each PE, but changes the persist unit to include the MT nodes received dirty from other PEs.

The persist unit tracks the changes to the cached security metadata by tracking the dirty cached nodes and limiting this tracking to the address only for encryption counters [12]. In case of a crash, one of the PEs orchestrates the recovery process (assuming PE0) to prevent memory throttling. The recovery manager (PE0) starts recovering its security metadata cache by reading its mirror region, recovering the lost encryption counters, fetching the lost intermediate nodes, and

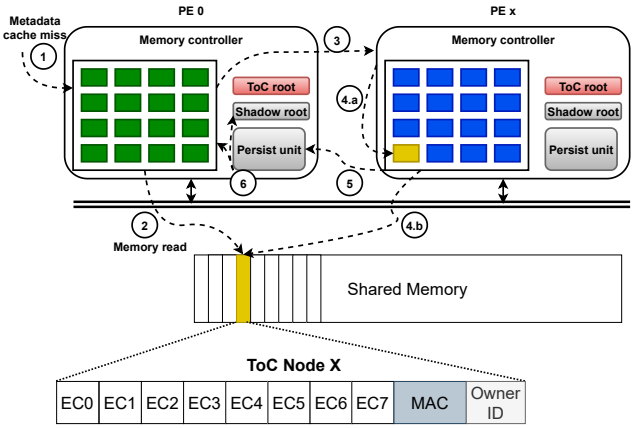

Fig. 6. Minerva's request for a cacheline.

finally verifying the integrity of the restored security metadata cache. Once done, PE0 sends an initiate-recovery to the next PE, which follows the same procedure to recover its cache and sends a recovery-done signal back to PE0. The process continues until all the PEs in the system finish the recovery. In case one of the PEs fail to recover, a recovery failure signal is raised and tampering is detected. Note that, the owner-ID bits are not included in the MAC calculations and therefore are not recovered. Thus, to avoid unnecessary broadcasts, during the recovery each PE ensures the correctness of the owner-ID bits of its cached nodes. The main difference between the persist unit and Phoenix [12] scheme is in tracking the nodes received dirty from other PEs, as Phoenix is a single PE scheme, it does not require tracking any newly fetched node.

3) Minerva's Operation: As Minerva's main goal is to reduce the performance overhead of maintaining security metadata cache coherence, it is only affected by the cache operations, which are the *update*, *evict*, and *misses*.

Updating a security metadata cacheline is done whenever a data cacheline is written back to memory. Since Minerva implements the lazy-invalidate scheme, it only needs to update the associated EC for each memory write and there will be no need to notify any other PE. However, as the ToC root is kept in a persistent register in each PE, the root updates are broadcasted to all PEs.

Misses are resolved by requesting the required ToC node from the component currently caching it. As shown in Figure 6, whenever a PE suffers a security metadata cache miss, it starts by reading the requested node from the memory to obtain the owner-ID as in steps 1 and 2. In step 3, the requester sends a request to the owner, which updates the requester's cacheline owner-ID, invalidates the requested cacheline from its cache and sends it to the requester in steps 4.a, 4.b, and 5. Finally, the requester updates its persist unit and insert the requested cacheline into the security metadata cache in step 6. Note that multiple PEs can request the node from its owner at the same time, which results in data races. Thus, Minerva locks the requested ToC node once a request for the node is received (step 4.a) and does not unlock the node until received by the requester (step 6).

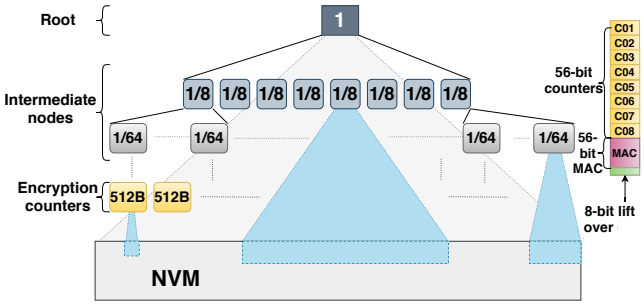

Fig. 7. Memory coverage of ToC nodes.

Evicting a security metadata cacheline in Minerva is perceived by the way it was initiated. Since Minerva is a system-level scheme, it differentiates between evicting a cacheline due to replacement, and evicting a cacheline due to ownership transfer request. Evicting a cacheline due to replacement requires updating its immediate parent node as it will be written back to the memory. On the other hand, transferring the cacheline to another PE does not require updating anything except for the owner-ID bits, as discussed earlier.

IV. Methodology

We modeled Minerva, CCCM-Update, and CCCM-Invalidate security metadata coherence schemes in the Structural Simulation Toolkit (SST) simulator [32]. To accurately model our work, we modified the memory controller to handle the security metadata, and implemented the coherence messaging exchange scheme, the security metadata caches, and the message communication between PEs. The configurations of the simulated PEs and the memory system are shown in Table I.

TABLE I
Configurations of the simulated system.

System Configuration	
Number of PEs	2, 4, 8, 16
Fabric latency	40ns
Fabric Attached Memory	
Technology	PCM-based NVM
Capacity	16GB
Latencies	Read 60ns, Write 150ns
Processing Element (PE)	
Processor	4 Cores, x86-64, Out-of-Order, 2.00GHz
L1 Cache	Private, 4 Cycles, 32KB,8-Way
L2 Cache	Private, 6 Cycles, 256KB, 8-Way
L3 Cache	Shared, 12 Cycles, 1MB/core, 16-Way
Cacheline Size	64Byte
Encryption Parameters	
Security Metadata Cache	256KB, 8-Way, 64B Block
Interconnect Parameters	
Topology	Torus
Ring BW, I/O Buffers	96 GB/s, 4KB
MMU	Opal [23]

To evaluate our proposed scheme, we ran combinations of the persistent applications shown in Table II, similar to the previous work in [11], [30]. Each PE is running a different

TABLE II
WORKLOADS.

Workload	Benchmarks	Workload	Benchmarks
Mix1	libquantum	Mix8	arswp, btree
Mix2	libquantum, soplex	Mix9	hashmap, randwr
Mix3	lulesh2.0, lbm	Mix10	randwr, rbtree
Mix4	lulesh2.0	Mix11	seqwr, arswp
Mix5	miniFe.x	Mix12	seqwr, hashmap
Mix6	soplex. lbm	Mix13	tpcc, btree
Mix7	soplex	Mix14	tpcc, rbtree

benchmark for 500M instructions. In our model, we connected all the PEs to share the fabric attached memory, where each PE has a cache hierarchy as described in Table I. Additionally, the persistent applications expect to recover from crashes and thus the application's data need to be NVM resident, which requires data writes to be flushed all the way to the NVM. Consequently, the security metadata of the NVM need to be updated with each write.

V. EVALUATION

In this Section, we discuss the evaluation results of the CCCM and Minerva schemes against an ideal coherency scheme that incurs no overheads.

A. Minerva's Impact on ToC Accesses

Minerva implements a lazy-invalidate scheme that reduces the accesses to upper MT levels, which results in less frequent communication of MT updates. Figure 8 shows Minerva's accesses to the ToC levels compared to other schemes. We observe that Minerva shows a similar behaviour as the ideal scheme, but generating 18% more accesses to the ToC than the ideal scheme. As shown in the figure, Minerva serves 65.8% of the requests from the first ToC level, 10% from each level of levels 2-4, 3.5% from level5, and less than 0.3% from upper levels. Thus, Minerva only requires to request the security metadata cacheline's ownership in a very small percentage of the requests.

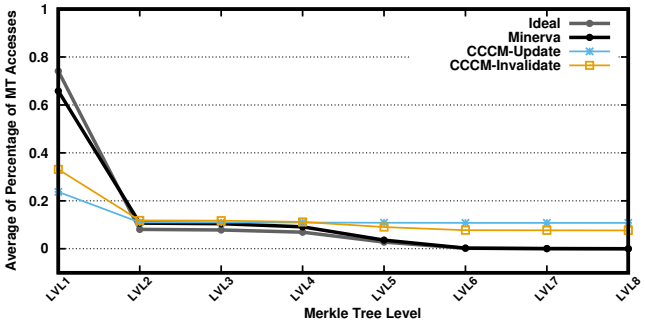

Fig. 8. Percentage of accesses to different MT levels.

In contrast, the CCCM schemes generate 3.16x and 2.21x for the invalidate and update schemes, respectively. We observe that majority of the ToC requests in the update scheme are caused by updating all the ToC levels, and the invalidate scheme requests are generated because of invalidating the nodes of the upper ToC levels. As shown in Figure 8, the update scheme is serving 23.7%, and the invalidate scheme is serving 33% from the first level. Upper levels are serving 11.8%-7.7% each. Therefore, these updates are causing the high overheads of the CCCM schemes.

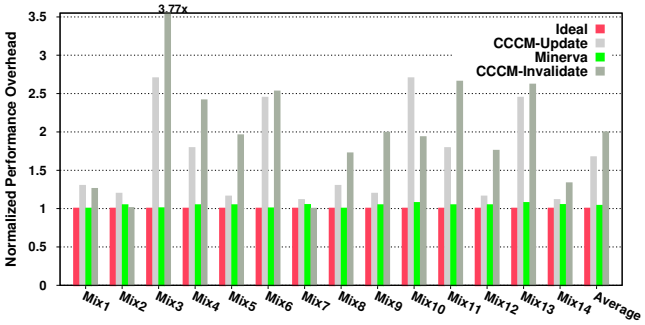

Fig. 9. Minerva's impact on performance.

B. Minerva's Impact on Performance

As shown in Figure 9, Minerva has a very low performance overhead as it has an average performance overhead of 3.8%, CCCM-Update has 67.1%, and CCCM-Invalidate has 99.5%. The performance overhead of the CCCM schemes is a function of data cache's miss rate (dirty data evictions). For instance, Mix3, Mix6, Mix10 and Mix16 have the lowest data cache's hit rates of 87-89%. However, Mix6 performs better as half of the PEs are executing the *soplex* benchmark, which has a low MPKI, reducing the frequency of sending coherence messages to invalidate/update security metadata cachelines in other PEs. The performance of other Mixes is a direct function of the data cache's hit rate. We observe that CCCM-Update scheme always performs better than CCCM-Invalidate scheme, which is caused by the frequent invalidation of the security metadata cachelines, resulting in memory reads.

In contrast, Minerva's performance is a function of the security metadata cache's miss rate. Thus, workloads with the highest security metadata cache's miss rates incur the highest performance overhead. As shown in Figure 9, Mix10 and Mix13 are incurring the highest performance overhead in Minerva, which is caused by the low security metadata cache's hit rate, as shown in Figure 10. Note that Minerva's performance overhead is minimal, since the security metadata misses are actually a fraction of the data caches' misses. With an average data caches' hit rate of 95% and an average security metadata cache's hit rate of 92.1%, Minerva only needs to request security metadata cachelines from other PEs on 0.3% of the memory accesses.

C. Minerva's Impact on Number of Writes

Figure 11 shows the normalized memory writes incurred by Minerva. The number of writes for CCCM-Update and the CCCM-Invalidation schemes are not shown as it is **8x**, due to the need to persist the whole ToC path on each update.

In comparison, Minerva increases the average number of shared memory writes only by 20.8%. Minerva updates the

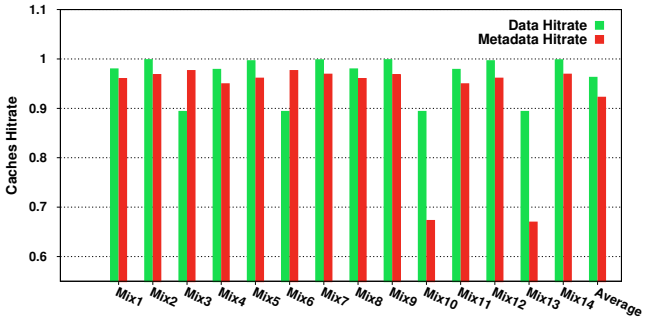

Fig. 10. Caches hitrate.

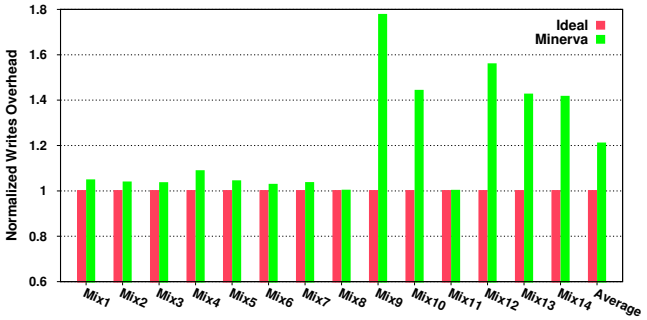

Fig. 11. Minerva's impact on the number of writes.

ToC node's owner-ID in the memory for each security metadata cache's miss. Therefore, workloads with high security metadata miss rates are showing high overheads. However, Mix13 and Mix14 are showing a high increase in the number of writes, which is caused by *TPCC* benchmark that is a read intensive and has a low number of writes. Therefore, the owner-ID updates are showing a high increase in the writes overhead. We Observe that Mix9 and Mix12 are showing the highest increase in the number of writes, which is caused by the frequent owner-ID updates. Note that Minerva's write overhead is a direct function of the security metadata cache's hit rate. Therefore, workloads with the lowest security metadata hit rates should incur the highest increase in the number of writes. However, the impact of this increase is minimal as Minerva incurs an extra write for each security metadata cache miss, which is less than 5% for most of the applications. Additionally, when the number of writes is normalized to the application's number of writes, the normalized value can be unrepresentative especially in read intensive applications.

D. Minerva's Impact on Number of Reads

Figure 12 shows the normalized number of memory reads for the evaluated schemes. CCCM-Update has an average read overhead of 0.26% caused by receiving and caching security metadata from other PEs, which causes few evictions of the required metadata. The CCCM-Invalidation scheme has an average of 519.9% number of reads, which is caused by invalidating the upper ToC levels. On the other hand, Minerva has an average reads overhead of 10.7% which is caused by reading the owner-ID of the requested node before sending the request to the owner. Mix10 and Mix13 are showing the highest increase in the number of reads, which is caused by the low security metadata cache's hit rate for these mixes.

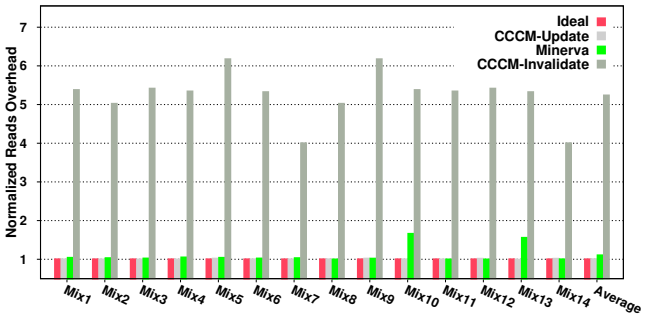

Fig. 12. Minerva's impact on the number of reads.

E. Minerva's Impact on Traffic

Figure 13 shows the exchanged packets in the system in different schemes. The CCCM-Invalidation scheme has an average of 420.6% extra packets, caused by the metadata reads, metadata writes, invalidation messages, and acknowledgments. The CCCM-Update scheme results in an average of 450.1% extra packets caused by the metadata writes, update messages, and acknowledgments.

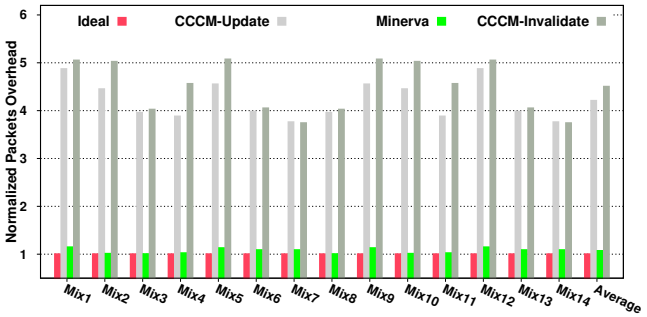

Fig. 13. Minerva's impact on traffic.

Note that the CCCM-Invalidation scheme only requires to send the updated encryption counter to invalidate the whole branch. On the other hand, the CCCM-Update would require sending the whole branch to prevent multiple reads by each PE to update its cached security metadata nodes. However, both schemes receive a single acknowledgment message. Minerva has an average of 12.1% extra packets caused by the broadcasts to obtain the memory owned nodes, as most of the misses come from encryption counters that are not used by any other PE as the applications are not sharing data.

F. Sensitivity Analysis on Scalability

We analyze how Minerva performs with increased number of PEs in the system (from 2 to 16), and our experimental results show that Minerva barely affects the performance as shown in Figure 14. Each performance metric is normalized to ideal memory encryption and integrity verification scheme

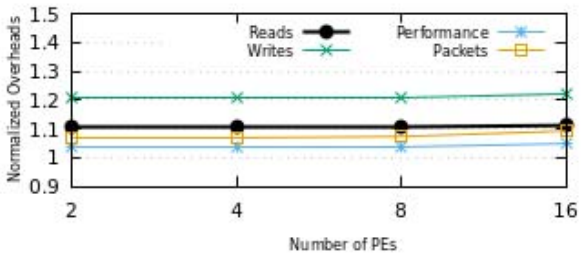

Fig. 14. Minerva sensitivity vs number of PEs.

with the same number of PEs, that assumes coherence without sending any coherence messages.

We observe that Minerva's performance stays the same when 4-8 PEs are used, but increases by 1.1% when 16 PEs are used. This small increase is caused by the increase in the broadcast range. The number of memory reads increased by 0.6% when 16 PEs are used, but stays the same for less than 16 PEs. The number of transferred packets increased by 0.4% when scaled to 8 PEs, and increased by 2.3% for 16 PEs due to the broadcasts for memory owned nodes. Note that increasing the number of PEs has two contradicting effects on Minerva's performance. On one side, increasing the number of PEs increases the broadcast range when the root is updated or a memory owned node is requested. On the other side, increasing the number of PEs increases the percentage of PEs owned security metadata, which reduces the number of broadcasts. The writes overhead stays the same for 4-8 PEs, but increased slightly by 1.2% for 16 PEs, which is caused by the owner-ID updates. These result are expected, as Minerva enables secure FAM architectures by adding a single message to the security metadata node's owner, followed by a single write to update the ToC node owner.

VI. Related work

The most related works to Minerva are the secure memory in Non-Uniform Memory Access (NUMA) architecture [34], Anubis [41], Phoenix [12], Triad-NVM [14], and Osiris [40]. The proposed scheme in [34] addresses the security requirements for distributed shared memory systems in NUMA architectures. In the proposed solution, each processor handles its memory security and the security metadata of one processor memory cannot be cached by other processors. Whenever any node tries to access another nodes' memory, it has to go through the home processor for that memory. Anubis [41] targets the recovery time problem of the encrypted and integrity-protected NVM. Anubis allocates a memory region in the NVM to persist the cache updates, and then uses the allocated region to recover the cache content after a crash. Phoenix [12] is an optimized version of Anubis [41], which reduces the number of writes to the NVM by adding more to the recovery time. Triad-NVM [14] discusses the performance and recovery time trade-off and suggests persisting N-levels of the BMT to reduce the recovery time. However, Triad-NVM does not work with ToC. Osiris [40] targets the encryption counters recovery problem and implements a stop-loss mechanism to persist the encryption counters on each N-th write. Osiris also relies on the ECC bits as a sanity check to recover the lost counters. The memory encryption engine [20] describes the details of the memory encryption engine of Intel's SGX, and describes the details of the ToC.

Several works were done in the NVM security and persistency community [35]–[37]. Morphable Counters [35] pack more encryption counters in a cacheline, thus increasing the cacheability and boosts the performance. Vault [37] proposes a variable arity integrity tree to reduce the tree size and improve the cacheability. However, the majority of the works discuss the performance overhead of NVM encryption and integrity verification, and propose optimizations to reduce these overheads. Caching Techniques [10] proposed a split-tree scheme to protect the integrity of the global memory in FAM architecture. However, the proposed technique addresses the problem of a single PE and does not address the security metadata coherence problem in multiple PEs.

VII. Conclusion

Minerva is a novel memory controller design that solves the security metadata coherency problem in small to medium scale FAM architectures. Unlike traditional coherence mechanisms, Minerva uses a lazy-invalidate scheme that limits the update to the encryption counter. Additionally, Minerva reduces the performance overhead significantly due to the exclusive caching, which makes the coherence messages a function of the security metadata cache misses instead of security metadata cache updates. In summary, Minerva achieves security metadata coherence with a performance overhead of 4.8% for systems with up to 16 PEs, and reduces the number of extra writes to the NVM to 20.9%.

VIII. Acknowledgments

This paper describes objective technical results and analysis. This research was partially 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, U.S. Department of Energy, or the U.S. Government. Approved for public release. Distribution is unlimited.

References

[1] AMD Memory Encryption. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf. Accessed: 2021-08-13.
[2] CCIX specifications V 1.0. https://www.ccixconsortium.com/library/specification/. Accessed: 2021-08-13.
[3] CXL 1.1 specifications. https://www.computeexpresslink.org/. Accessed: 2021-08-13.
[4] Gen-z overview. https://genzconsortium.org/wp-content/uploads/2019/11/Gen-Z-Security.pdf. Accessed: 2021-04-06.
[5] Gen-z security. https://genzconsortium.org/wp-content/uploads/2019/04/Gen-Z-Overview-2019.pdf. Accessed: 2021-04-06.
[6] GenZ specifications kernel description. https://genzconsortium.org/faqs/. Accessed: 2019-09-30.

[7] Intel Architecture, Memory Encryption Technologies Specification. https://software.intel.com/sites/default/files/managed/a5/16/Multi-Key-Total-Memory-Encryption-Spec.pdf. Accessed: 2021-08-13.

[8] Intel Promises Full Memory Encryption. https://hardware.slashdot.org/story/20/02/29/0433242/chasing-amd-intel-promises-full-memory-encryption -in-upcoming-cpus . Accessed: 2021-08-13.

[9] Intel® Optane™ DC Persistent Memory Operating Modes Explained. https://itpeernetwork.intel.com/intel-optane-dc-persistent-memory-operating-modes/gs.xtmcnw. Accessed: 2021-08-13.

[10] Mazen Alwadi and Amro Awad. Caching techniques for security metadata in integrity-protected fabric-attached memories. *EAI Endorsed Transactions on Security and Safety*, 7(24):e1, 2020.

[11] Mazen Alwadi, Vamsee Reddy Kommareddy, Clayton Hughes, Simon David Hammond, and Amro Awad. Stealth-persist: Architectural support for persistent applications in hybrid memory systems. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 139–152. IEEE, 2021.

[12] Mazen Alwadi, Aziz Mohaisen, and Amro Awad. Phoenix: Towards persistently secure, recoverable, and nvm friendly tree of counters, 2019.

[13] Mazen Alwadi, Aziz Mohaisen, and Amro Awad. Promt: optimizing integrity tree updates for write-intensive pages in secure nvms. In *Proceedings of the ACM International Conference on Supercomputing*, pages 479–490, 2021.

[14] Amro Awad, Yan Solihin, Laurent Njilla, Mao Ye, and Kazi Zubair. Triad-nvm: Persistency for integrity-protected and encrypted non-volatile memories. In *Proceedings of the 46th International Symposium on Computer Architecture*, pages 169–180. ACM, 2019.

[15] Amro Awad, Suboh Suboh, Mao Ye, Kazi Abu Zubair, and Mazen Al-Wadi. Persistently-secure processors: Challenges and opportunities for securing non-volatile memories. In *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 610–614, 2019.

[16] Amro Awad, Yipeng Wang, Deborah Shands, and Yan Solihin. Obfusmem: A low-overhead access obfuscation for trusted memories. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 107–119, 2017.

[17] Mark S Birrittella, Mark Debbage, Ram Huggahalli, James Kunz, Tom Lovett, Todd Rimmer, Keith D Underwood, and Robert C Zak. Intel® omni-path architecture: Enabling scalable, high performance fabrics. In *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*, pages 1–9. IEEE, 2015.

[18] Victor Costan and Srinivas Devadas. Intel sgx explained. *IACR Cryptology ePrint Archive*, 2016(086):1–118, 2016.

[19] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G Shin. Efficient memory disaggregation with infiniswap. In *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, pages 649–667, 2017.

[20] Shay Gueron. A memory encryption engine suitable for general purpose processors. 2016. https://eprint.iacr.org/2016/204.

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

[22] Kimberly Keeton. The machine: An architecture for memory-centric computing. In *Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*, volume 10, 2015.

[23] Vamsee Reddy Kommareddy, Amro Awad, Clayton Hughes, and Simon David Hammond. Exploring allocation policies in disaggregated non-volatile memories. In *Proceedings of the Workshop on Memory Centric High Performance Computing*, pages 58–66, 2018.

[24] Vamsee Reddy Kommareddy, Simon David Hammond, Clayton Hughes, Ahmad Samih, and Amro Awad. Page migration support for disaggregated non-volatile memories. In *Proceedings of the International Symposium on Memory Systems*, pages 417–427, 2019.

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

[26] Benjamin C Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger. Phase-change technology and the future of main memory. *IEEE micro*, (1):143–143, 2010.

[27] Zhongqi Li, Ruijin Zhou, and Tao Li. Exploring high-performance and energy proportional interface for phase change memory systems. *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 210–221, 2013.

[28] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K Reinhardt, and Thomas F Wenisch. Disaggregated memory for expansion and sharing in blade servers. *ACM SIGARCH computer architecture news*, 37(3):267–278, 2009.

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

[30] Sihang Liu, Korakit Seemakhupt, Gennady Pekhimenko, Aasheesh Kolli, and Samira Khan. Janus: Optimizing memory and storage support for non-volatile memory systems. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, pages 143–156. IEEE, 2019.

[31] Thomas S Messerges. Securing the aes finalists against power analysis attacks. In *International Workshop on Fast Software Encryption*, pages 150–164. Springer, 2000.

[32] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, Rolf Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. The structural simulation toolkit. *SIGMETRICS Performance Evaluation Review*, 38(4):37–42, 2011.

[33] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 183–196. IEEE Computer Society, 2007.

[34] Brian Rogers, Chenyu Yan, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Single-level integrity and confidentiality protection for distributed shared memory multiprocessors. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 161–172. IEEE, 2008.

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

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

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

[38] Jason Taylor. Facebook's data center infrastructure: Open compute, disaggregated rack, and beyond. In *Optical Fiber Communication Conference*, pages W1D–5. Optical Society of America, 2015.

[39] Yao Wang, Andrew Ferraiuolo, and G Edward Suh. Timing channel protection for a shared memory controller. In *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pages 225–236. IEEE, 2014.

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

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