SecDDR: Enabling Low-Cost Secure Memories by Protecting the DDR Interface

Ali Fakhrzadehgan
The University of Texas at Austin
alifakhrzadehgan@utexas.edu

Prakash Ramrakhyani
Arm
prakash.ramrakhyani@arm.com

Moinuddin K. Qureshi
Georgia Tech
moin@gatech.edu

Mattan Erez
The University of Texas at Austin
mattan.erez@utexas.edu

Abstract—The security goals of cloud providers and users include memory confidentiality and integrity, which requires implementing replay attack protection (RAP). RAP can be achieved using integrity trees or mutually authenticated channels. Integrity trees incur significant performance overheads and are impractical for protecting large memories. Mutually authenticated channels have been proposed only for packetized memory interfaces that address only a very small niche domain, require fundamental changes to memory system architecture, and assume fully-trusted modules. We propose *SecDDR*, a low-cost RAP that targets direct-attached memories, like DDRx. SecDDR avoids memory-side data authentication, and thus, only adds a small amount of logic to memory components and does not change the underlying DDR protocol, making it practical for widespread adoption. In contrast to prior mutual authentication proposals, which require trusting the entire memory module, SecDDR targets untrusted modules by placing its limited security logic on the DRAM die (or package) of the ECC chip. Our evaluation shows that SecDDR performs within 1% of an encryption-only memory without RAP and that SecDDR provides 18.8% and 7.8% average performance improvements (up to 190.4% and 24.8%) relative to a 64-ary integrity tree and an authenticated channel, respectively.

Index Terms—Memory security, Replay attacks, Memory integrity

I. INTRODUCTION

Trusted data-center infrastructure is crucial for users to move their applications and data to the cloud. One risk is attacks on main memory that have been demonstrated for accessing private data [17], [27], [28], [55] and even for taking over entire servers [24], [48]. To mitigate against main memory vulnerabilities and physical attacks, *trusted execution environments* (TEE), such as Intel Software Guard Extensions (SGX) [15], provide secure off-chip memory that ensures data confidentiality and integrity.

Securing memory incurs application slowdown because each memory access requires additional security metadata accesses. Of particular interest to this paper is that for integrity protection, each data block is guarded by a cryptographic *message authentication code* (MAC), which is stored with the data in the memory. The processor has to fetch the stored MAC to verify data integrity. In this paper, we focus solely on reducing the memory integrity overheads and rely on unmodified prevalent confidentiality schemes [3], [15], [18].

The MAC itself must also be protected to provide *complete* integrity guarantees and prevent *replay attacks*. In a replay attack, the attacker bypasses the integrity verification by replaying an *older* pair of data and its MAC (e.g., a 72-byte tuple for 64-byte data and an 8-byte MAC). This pair appears

to be correct on the processor, however, it is stale and may corrupt execution. For *replay attack protection* (RAP), secure processors may create an integrity tree over the MACs or over the encryption counters [15], [44]. The processor traverses the tree from the leaf to the root to verify the integrity and freshness of the data or counters. The root of the tree is always stored on chip and cannot be tampered with. This integrity tree increases memory bandwidth pressure and access latency as it requires several additional accesses to traverse.

While prior work has proposed techniques to lower the cost of MACs [10], [46] and other security metadata [57], integrity trees continue to limit scalability and performance of secure memories. This is because the tree traversal overhead is proportional to its size and height, which depends on the protected memory size. Applications with large memory footprints experience a significant slowdown due to either expensive tree walks or the extra data movement caused by the numerous page faults required to manage the small efficiently-protected memory space afforded by small integrity trees [53].

Prior work, such as compact high-arity trees [45], [53], has had only limited success in addressing this harsh tradeoff between per-access integrity tree overhead and a small protected memory. This crucial limitation of integrity trees continues to be a major obstacle to widespread commercial adoption of complete memory protection. For example, while new products have extended memory encryption to the entire memory space (e.g., Total Memory Encryption (TME) [18] and Secure Encrypted Virtualization (SEV) [3]), replay attack protection is either missing, or is restricted to only a small portion of memory (e.g., 96MB for Intel SGX and a small portion of memory in Apple's Secure Enclave Processor [4]).

We propose *SecDDR* to protect the DDR interface against practical replay attacks at much lower cost than current industrial and academic approaches. SecDDR uses a narrow secure channel to *encrypt* the MAC (E-MAC) and protect it while data is transferred between the processor and memory. This prevents an attacker from replaying a stale *(Data, MAC)* pair as the plain-text MAC is not observable. The channel counters are not stored and are incremented at each memory transaction, making E-MACs temporally unique such that an E-MAC is never repeated with its data. MACs are stored un-encrypted in memory, protecting the integrity of the data at rest. SecDDR performs MAC verification *only* on the processor.

The E-MACs fully guarantee data integrity, but are vulnerable to a stale-data attack where the attacker manipulates the command and address signals to force a memory write

to not reach its destination address. The old *(Data, MAC)* are returned when that address is read again, providing a stale pair. We protect against such attacks by introducing *encrypted write cyclic redundancy code (CRC)* that extends the *extended write CRC (eWCRC)* approach of All-Inclusive ECC (AI-ECC) [23] to allow the memory device to identify mismatched addresses and data before performing the write, thus detecting any tampering. Like E-MACs, we encrypt the eWCRC to both prevent an attacker from choosing values that can still pass the non-cryptographic CRC check and to prevent new information leakage.

Mutually authenticated channels between the processor and memory have also been proposed to defeat replay attacks without an integrity tree. InvisiMem [2] applies this to the packetized protocol of the Hybrid Memory Cube (HMC) [37]. However, direct adaptation of InvisiMem to DDRx dual in-line memory modules (DIMM) is impractical. First, the security guarantees of InvisiMem (and any mutual authenticated channel) require that the entire DIMM be trusted. This is acceptable for the logic and memory layers in an HMC, but does not hold true for commodity DIMM-style modules that comprise multiple discrete components. One could extend InvisiMem's trusted computing base (TCB) to include the entire module, however, this leaves the system vulnerable to physical attacks on the DIMM. Moreover, mutual authentication for DDRx requires fundamental changes as DDRx is not packetized, has strict standardized timing parameters, and commodity DIMMs do not have a centralized data buffer in which mutual authentication can be computed (Section VI).

We develop SecDDR to overcome the challenges of providing a low-cost and scalable RAP for commodity DDRx modules. In contrast with prior work [2], which its successful adoption *requires* trusting the entire memory module, SecDDR can be easily tailored for untrusted DIMMs (as well as trusted) with negligible performance overhead, eliminating the vulnerability to on-DIMM physical attacks and malicious units. To this end, we place SecDDR's limited security logic in some of the DRAM chips (the ECC chips). While implementing this logic on the DRAM die is costly, it is practical considering advancements in logic-in-memory technologies demonstrated by DDR5 on-die ECC and recent industrial processing-in-memory prototypes [26], [29], [30], [40]. We anticipate this to be a boon to memory vendors as the market for server memory is large, as well as to processor vendors who can offer highly-secure memory with less overhead.

Overall, this paper makes the following contributions:

- We analyze different replay attack scenarios and observe that replay attacks can be mitigated by protecting only the MACs as they traverse the memory channel.
- We propose SecDDR, a low-cost replay attack protection mechanism for the DDRx standard. SecDDR uses dedicated encryption units to encrypt MACs, protecting them on the bus, and synchronized channel encryption counters to protect against data-at-rest attacks.
- To protect against on-DIMM vulnerabilities, we develop SecDDR for untrusted DIMMs, including address manip-

ulation and man-in-the-middle attacks. We further discuss how SecDDR is compatible with trusted DIMMs as well.
- SecDDR enables low-overhead integrity protection while supporting both counter-mode encryption and recent commercial approaches that forego counters for the AES-XTS encryption scheme [3], [18]. We show that AES-XTS provides a substantial performance boost over counter-mode encryption and is compatible with SecDDR but not with state-of-the-art integrity-tree designs.
- We evaluate SecDDR and show that it provides 18.8% and 7.8% average performance improvements (up to 190.4% and 24.8%) relative to a 64-ary integrity tree and an authenticated channel based on InvisiMem, respectively. SecDDR performs within 1% and 3% of encrypt-only memories with AES-XTS and AES-CNT, respectively.

II. BACKGROUND & MOTIVATION

A. Threat Model

We consider a threat model similar to SGX [8], [15]. The software that runs in the secure environment (e.g., the enclave) is the only software part of the Trusted Computing Base (TCB). Other processes (including the OS and the Hypervisor) are untrusted and are restricted with a hardware-based isolation. An adversary can perform passive (eavesdropping on application information) or active (tampering with the data) physical attacks. The processor chip is part of the TCB and cannot be tampered with. The attacker can target any off-chip component, including the memory bus and DIMMs.

We consider a modern DDR4/5 module architecture (DIMM) to cover the attack surface of a memory module. A memory module is composed of several DRAM chips. Each chip has a narrow interface (e.g., 4, 8, or 16 bits). To create a wider data bus, multiple chips are organized in groups called *ranks*, all operating in lockstep within a rank. A module can have multiple ranks for higher capacity.

The large number of memory chips on a high-capacity DIMM increases the capacitive load on the memory bus and the module's interconnects, which adversely impacts signal stability and integrity. To mitigate this problem, industry has adopted *registered DIMMs (RDIMM)* and *load-reduced DIMMs (LRDIMM)*. In these designs, the I/O signals to each of the DRAM chips are decoupled by adding extra buffer chips to the module [43], [54]. Buffer chips include a single centralized registered clock driver (RCD) chip for the command, control, clock, and address (CCCA) signals, and several distributed data buffers (DB) for buffering the data pins. Whereas RDIMMs only have the RCD to buffer the CCCA, LRDIMMs have both RCD and distributed DBs to buffer both CCCA and data.

In line with prior work [2], [51], we consider attacks that target the interconnects on the DIMM, but keep physical attacks on internal circuits within a package, such as the processor chip, DBs, RCD, and the DRAM chips out of scope. In-package attacks are significantly harder to perform as they require successfully desoldering packages, removing different transistor layers to reach the target cells or connections, and tapping circuits that are at micron/nanometer scale while

maintaining high-performance operation within a running system. We do not consider address/command traffic, bus utilization, power, and electromagnetic side-channels because these are confidentiality issues. This is out of the scope of this paper as our focus is integrity protection and our approach does not affect the confidentiality mechanisms and SecDDR does not open additional side channels (see Section III-D).

B. Secure Memory Basics

Ensuring the off-chip data security has two aspects: confidentiality and integrity. Confidentiality is needed to protect data *privacy*. Integrity is needed to protect the *correctness* of the off-chip data, i.e., ensuring that data has been indeed written by the trusted software running on the trusted processor and has not been modified by an adversary in the interim.

Data Confidentiality. Secure processors use *encryption* to ensure data confidentiality. Intel SGX [15] uses counter-mode encryption, in which each cache-line is associated with an encryption counter that is stored off-chip. Recent products (e.g., Intel TME [18] and AMD SEV [3]) have managed to extend memory encryption to the entire memory space by adopting low-cost XOR-Encrypt-XOR (XEX) encryption mode (e.g., AES-XTS) [56], omitting the encryption-counter storage and memory bandwidth overheads.

Data Integrity. To protect data integrity, each cache-line is guarded by a *message authentication code* (MAC), which can detect arbitrary data modifications. The MACs are stored with the data in memory and need to be fetched to verify data integrity, which incurs storage and memory bandwidth overheads. To provide low-cost integrity protection, recent products (e.g., Intel TDX [19], [20]) and academic proposals (e.g., SafeGuard [10]) place both MAC and error correction code (ECC) in the ECC chips and transfer them using the ECC portion of the bus. This eliminates the storage and bandwidth overheads of the MACs while maintaining ECC protection [10].

In this paper, we consider a baseline system equipped with similar low-cost confidentiality and integrity mechanisms. Unfortunately, MACs alone cannot provide *complete* integrity protection as they must also be protected to prevent *replay attacks*. In the next section, we discuss how a replay attack is performed and explain current mitigation techniques.

C. Replay Attacks & Defenses

In this section, we formally define replay attacks and describe how they can bypass integrity checks. We explain which data is vulnerable to replay attacks and discuss the existing mitigations.

1) How to Perform a Replay Attack?

A replay attack can bypass integrity verification if it does not result in a MAC mismatch. Any corruption in the data or its MAC causes an integrity verification failure with sufficiently high probability, except when both data and its MAC are replayed at once. In other words, if $(c, m)_{t_0}^a$ is the state of the cache-line c at address a with its MAC m at time t_0, and $(c, m)_{t_1}^a$ is the state at time $t_1 > t_0$, overwriting the tuple $(c, m)_{t_1}^a$ with $(c, m)_{t_0}^a$ would pass the MAC verification of

the line c. Note that it is important to replay the tuple to the *same address* since physical addresses are included in the MAC [15], [57]. Thus, the attacker has to precisely track memory addresses, memoize changes to a specific location over time, and precisely replay a $(Data, MAC)$ tuple to avoid signaling an integrity violation. Figure 1 depicts logical view of a replay attack on address a. Integrity verification passes at time t_2.
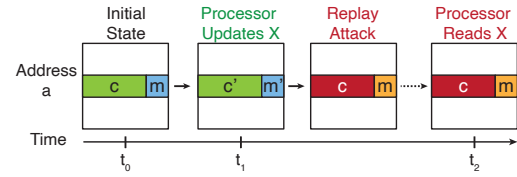

Fig. 1: Logical view of replay attack on address a.

2) Which Data is Vulnerable to a Replay Attack?

We categorize different types of replay attacks based on whether they are done on data at rest or data in motion.

Data at Rest. Off-chip data *at rest* is the data that is stored in memory that the application is not currently operating on. TEEs protect secure environments (enclaves) using hardware-based isolation mechanisms [8] so that different processes (including the OS) cannot access each other's data. Thus, a software-based replay attack could not succeed.

The attacker could attempt to replay the data *indirectly* by inducing bit-flips (e.g., via Row-Hammer [24] or causing soft errors), however, we consider this type of attack impractical. Theoretically, it is not impossible to perform a replay attack by bit-flips, however, the likelihood of success would be extremely low as the attack needs to flip enough bits such that $(Data, MAC)$ match precisely. All demonstrated Row-Hammer attacks only induce a few bit-flips per-line (fewer than 10) [13], [22]. We do not know of any real-world replay attacks that have occurred using these means.

The attacker can target data at rest using *DIMM substitution*. The attacker keeps a version of the data by removing the DIMM and replaying the application state by plugging in that DIMM later. This attack relies on the *data remanence* effect [17].[1]

Data in Motion. Off-chip data *in motion* is data that is being transferred between the memory and the processor, such as an LLC fill or write-back. The replay attack is a *Man-In-The-Middle* attack, where the attacker either interposes traffic on the bus between the processor and the memory module or uses a malicious DIMM that is capable of analyzing and intercepting on-DIMM interconnect (e.g., via a trojan).

3) Current Defenses Against Replay Attacks

Integrity Trees. To defeat replay attacks, secure processors create an integrity tree over the MACs [12] or over the encryption counters [15], [44]. The processor traverses the tree

---

[1]Theoretically, the attacker can detach the memory chips from the module PCB and use a different PCB to replay the data. While this attack is possible on a module assumed to be trusted with no on-DIMM protection, SecDDR defeats this attack, as we describe in Section III-E.

from the leaf to the root to verify the integrity and freshness of the data or counters. The root is always stored on-chip and cannot be tampered with. For more details on tree designs and traversal refer to prior work [8], [15], [44], [45], [53], [57].

Mutual Authentication. Mutually authenticated channels between the processor and memory defeat replay attacks without an integrity tree. Each memory transaction on the bus is protected using a unique and dedicated per-transaction MAC (MAC_t). A MAC_t is generated with a transaction secret key (K_t), the data, and a non-repeating nonce, such as a counter (C_t), which both ends (the processor and the DIMM) are equipped with. C_t is incremented at each transaction and always has the same value on both ends. At each transaction, the sender (processor on writes and DIMM on reads), uses C_t to compute MAC_t for the transmitted data, i.e., $MAC_t = H_{K_t}(Data, C_t)$. On the receiving end, the MAC is recomputed and compared against the received MAC_t to verify data integrity and freshness before being stored in memory or used in the processor. Because C_t is unique, replay attacks are detected.

D. Goal: Practical Replay Attack Protection

Ideally, replay attack protection should be scalable, low-cost, and provide complete protection. Integrity trees do not scale to large capacity memories. Prior attempts that create a mutually authenticated channel on the memory bus require trusting the entire module and use packetized protocols, which are not applicable to modern DDRx DIMM design constraints. Our goal is to develop a low-cost solution that meets these requirements. Our aim is to make this solution practical for widespread adoption, and applicable to contemporary DIMMs without modifying the underlying memory protocol.

III. SECDDR: LOW-COST REPLAY ATTACK PROTECTION

SecDDR is based on the insight that integrity can be provided by blocking replay attacks on the bus. In brief, SecDDR creates a replay-protected channel on the memory bus by modulating the MACs for the data that is in transfer on the bus, as summarized in Section III-A. Section III-B describes how SecDDR protects against attacks on CCCA signals that feed stale data to the processor. We discuss how SecDDR protects from DIMM-substitution attacks in Section III-C. We describe SecDDR's TCB in Section III-E. Section III-F describes system initialization and the attestation process.

A. Replay-Protected Bus Using E-MACs

Although a mutually authenticated bus (Section II-C) protects the integrity of data in motion, it is not sufficient to protect integrity of data at rest. We need to store a MAC with the data in memory to verify its correctness on subsequent accesses (as in SGX/TDX). However, both (MAC_t) and (C_t) must be stored together for later verification, as (C_t) is incremented dynamically with every direction. This has a very high 25% total storage overhead for 64-bit counters and MACs.

One alternative is to discard MAC_t and delegate integrity protection of the data at rest to the memory module. On each data write, after verifying MAC_t, the memory module

generates a new MAC and stores it with the data. On reads, the memory module first performs a MAC verification, and if this verification passes, it can then generate MAC_t and transmit the data on the bus. InvisiMem [2] uses this technique, however, adapting this approach requires trusting the entire DIMM and fundamental changes in the DDRx module architecture (see Section VI), making it impractical.
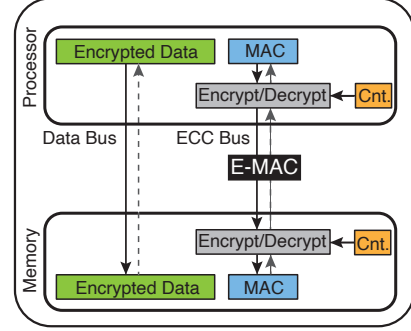

Fig. 2: SecDDR overview.

Given that replay attacks require bringing data into motion and that replay attack protection in mutual authentication is provided by making MAC_t *temporally unique*, we propose to eliminate memory-side integrity check via MAC encryption.

SecDDR uses _Encrypted MACs (E-MACs)_ to protect the bus. On a data write, the processor's memory encryption engine generates a MAC using $MAC = H_k(data, addr)$. However, before transferring this MAC on the bus, it is first encrypted to generate the E-MAC. Figure 2 shows an overview of SecDDR as it uses MACs to protect data at rest and repurposes them for protecting the data in motion. To generate the E-MAC, we XOR the MAC with a *one time pad* (OTP_t) generated using the transaction counter C_t. This effectively makes the MAC temporally unique and capable of detecting memory bus replay attacks (same as MAC_t in mutual authentication). The per-rank transaction counter is incremented at both the memory controller and memory module.

In SecDDR, *only* the processor performs integrity verification, and as a result, there is an important difference between how the processor and the DIMM use E-MACs. On each receiving end, we first XOR the E-MAC with the OTP_t to decrypt it and retrieve the original MAC. On the DIMM, this MAC is not used for verification and is simply stored, and the C_t discarded. Since this MAC is not verified on the DIMM, any attack at the time of write will remain undetected until the next read, just as with integrity trees. On the processor, this MAC is used for integrity verification, and a mismatch signals a failure, which could be due to multiple different sources:

- Bit-flips on the data-bus on a data read/write.
- Bit-flips while data was stored in the memory.
- Replay attack on the data read.
- Replay attack on a prior data write.

The processor cannot distinguish between different attack types, however, it can detect that *an* attack has occurred and that the data has been tampered with, which is what matters

for guaranteeing integrity. This is true since tampering with the E-MAC causes a wrong MAC to be computed after it is XOR-ed with the OTP_t. This is also true for write accesses, except the wrong MAC is stored, and its verification is deferred until the next read.

Compatibility with On-Die ECC. SecDDR provides replay attack protection by encrypting the MACs (E-MACs) to make them temporally unique. Although we have developed SecDDR based on state-of-the-art designs that place MACs in the rank-level ECC to mitigate MAC access overheads [10], [19], [46], MAC encryption is effective regardless of these optimizations.

B. Ensuring Command & Address Integrity

Attacks that we have considered so far are accomplished by directly targeting the data. However, the attacker can modify the CCCA signals to corrupt data integrity.

Attack Scenario. In SecDDR, any data corruption (including replay attacks) that happens at the time of write is not detected immediately and is deferred until the subsequent read. This method is safe only if the corrupted data is written *in place*, overriding the previous version of the $(Data, MAC)$ tuple. However, if the write is redirected to a different memory location, the stale $(Data, MAC)$ will remain in place, and the processor cannot tell that it is out of date.

Figure 3 shows an example in which the attacker creates such a scenario by corrupting the write address. Assume the processor reads the cache-line c at t_0, updates it to a new value c' at t_1, and attempts to read it at a later time t_2. In this case, between the time that the processor initially reads c and when it wants to write c', the DRAM row that c belongs to (row X) is closed. The memory controller has to first open row X, however, the attacker intercepts the *Activate* command and changes the row address to a different row (row Y). As a result, when the processor performs the write, c' will be written to the wrong row, leaving the original location with the stale $(Data, MAC)$. When the processor attempts to read the data at t_2, it opens row X reading the stale tuple, which passes MAC verification, completing the replay attack cycle.

In a similar attack, instead of corrupting the row address, the attacker can change the column address, writing c' to a different column in the original row. Note that if the processor ever reads the location that the attacker has redirected the writes to (i.e., row Y or the wrong column), SecDDR detects the attack as the line address is included in the MAC [15]. However, the attacker can orchestrate the attack in a way that remains undetected. Alternatively, the attacker can simply drop the write request instead of redirecting it to a different location. However, SecDDR can detect this case since dropping a request will cause a C_t mismatch between the processor and memory.

All-Inclusive ECC (AI-ECC) [23]. CCCA corruption can also happen from naturally occurring faults. AI-ECC mitigates these errors by extending the existing reliability measures of contemporary DRAM chips that protect data integrity to also protect the CCCA signals. For early-detection of write data transmission errors, DRAM chips use *write cyclic redundancy*
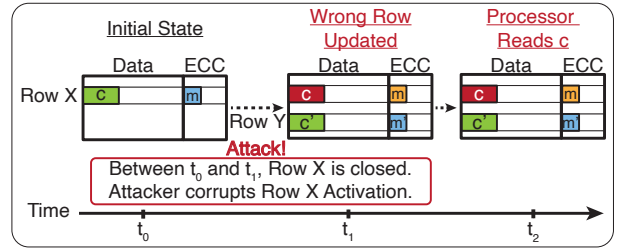

Fig. 3: Performing replay attack by corrupting the address bus. The attacker corrupts the row address of the Activate command when the processor is writing to cache-line c.

codes (WCRC) [35], [36], which are generated over the data transmitted to each chip. Enabling WCRC requires increasing the write burst length from 8 to 10 in DDR4 (16 to 18 in DDR5), in which the WCRC is transmitted to each chip over the last two beats (i.e., 16-bit WCRC with x8 device). Before storing the data, each DRAM chip internally recomputes the WCRC to make sure transmission was error-free. AI-ECC[2] proposes *extended write CRC (eWCRC)*, which enhances the WCRC to also include the *rank, bank, row,* and *column* address of the write to protect address bus integrity, as shown in Figure 4.
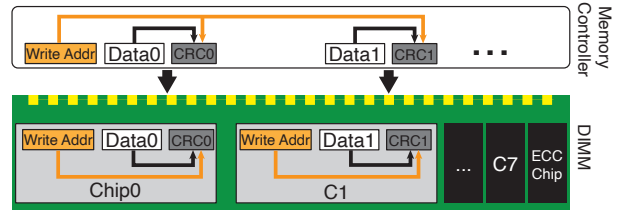

Fig. 4: AI-ECC's eWCRC [23]. The memory controller encodes the write address with the data in the WCRC. Each chip uses the address and data to verify the transaction.

SecDDR with Encrypted eWCRC. SecDDR defeats stale-data attacks from misdirected writes by enabling eWCRC and encrypting it, similarly to the E-MAC. The eWCRC of the ECC chip is generated before encrypting the MAC, and it is verified in the ECC chip after decryption of the E-MAC. However, because the eWCRC is not a cryptographic hash, the adversary can target specific bits in the message such that the corrupted eWCRC check would incorrectly pass. This is true even if the eWCRC is encrypted with the OTP_t used for the E-MAC. SecDDR therefore uses a separate OTP_t^w for write commands that uses the same key and transaction counter, but also includes the address used in eWCRC. This ensures that any corruptions to the address would flip numerous bits in the message and the eWCRC would detect the corruption. This approach increases the write latency because generating the OTP_t^w only starts after the write command is sent to the SecDDR DRAM chip and takes longer than tWCL. Importantly, the read latency is unaffected because the processor performs MAC verification.

---

[2]AI-ECC makes additional contributions to protect command and clock signals, however, these cases are detectable in an integrity-protected memory, and we do not further discuss them here.

An attempt to induce stale data by dropping a write transaction fails because C_t would not be incremented on the memory side, leading all following reads to fail verification. Finally, the attacker can potentially avoid updating a memory location by converting a write command to a read (and intercepting the response so the processor is not notified), which does not affect synchronization of the counters, and thus, remains undetectable. This attack can be defeated by simply using only even counter values for reads and odd counter values for writes so that command corruption results in a counter mismatch.

Security of Encrypted eWCRC. Assuming the worst-case bit error rate (BER) of 10^{-16} on the CCCA signals that is allowed by the JEDEC standard [36], channel transmission rate of $3200MTps$[3], and 26 CCCA and data signals for an x8 device [36], we expect to observe one CCCA error every 11.13 days per memory channel on average. Because the attacker only observes the eWCRC and MACs in their counter-encrypted form, *birthday attacks* on the eWCRC are not possible. In a brute force attack, each attempt has a success rate of 2^{-16} with the 16b eWCRC. Thus, even with a success probability of only 50%, the attacker must perform at least 4.5×10^4 attempts. Given that CCCA errors due to natural faults are rare and that a higher than-expected transmission error rate indicates an active attack, it takes 1,385 years to exhaust all trials on a single memory channel. In practice, the BER is much lower than the DDRx standard specifies, in the range of 10^{-22} to 10^{-21} [23], increasing the brute force attack duration to 138 million years. Even if the attacker launches a parallel attack on 1,000 nodes that each has 16 memory channels, the attack would still take more than 86,000 years.

C. DIMM-Substitution Attack Protection

Attack Scenario. An adversary can perform a replay attack via DIMM-substitution by taking advantage of the *data remanence* effect (Cold-boot Attacks [17]) to replay a victim application's state across boot or wake-up episodes. The attacker causes the system to crash or go to an idle state (i.e., DRAM self-refresh mode), takes away the DIMM, and freezes it to preserve the application state (and potentially copies it). After reboot or wake-up, the victim process continues execution as usual. At a later time, the attacker forces the system into crash/idle again. In the last step, instead of rebooting from the most recent state, the attacker uses the preserved *old* state, which forces the victim application to redo the *already-performed* computations, completing the replay attack cycle.

SecDDR's Efficacy Against the Attack. SecDDR defeats this type of attack by using the transaction counters (C_t). When the attacker tries to wake-up the system using the old state, it is improbable that C_t on the DIMM and the processor would match (the likelihood of a match is $\frac{1}{2^{64}}$), causing the OTP_t on the DIMM and the processor to be different (producing and retrieving different E-MACs), and the attack to fail. With a 64-bit C_t, we will not observe counter overflow in the system

lifetime, as it takes more than 500 years to cause an overflow, even assuming one transaction every nanosecond per rank.

Non-Adversarial DIMM Replacement. It is possible that a DIMM should be replaced for various legitimate reasons (e.g., system upgrade, faulty device) that must be differentiated from an attack. The difference between such cases and a DIMM-substitution attack is that the processor is explicitly notified of the replacement and expects to start from a *clean* state (as opposed to continuing from the previous architectural state in the memory). That is, any prior data in the memory should be discarded, by clearing the memory during boot or DIMM initialization (see Section III-F).

D. Vulnerability to Side-Channels

SecDDR does not introduce any new side-channels. All of the used cryptographic primitives have constant latency that is independent of the data; the latencies are either hidden from the access critical path or are equally imposed on all accesses. The encryption/decryption of E-MACs does not add any timing variation to reads, as the OTP_t is always pre-computed independently of transaction timing. The CRC logic also has constant latency. Thus, its extra latency and that of the longer write burst remains indistinguishable among different writes. With SecDDR, writes are slower than reads, however, this does not open a new physical side-channel as the command type and flow of traffic are already observable on the memory bus. From the software perspective, this is not a new side-channel as writes are already slower than reads because the memory controller prioritizes reads.

E. Trusted Computing Base for SecDDR

The processor chip is the only hardware component in SGX's TCB. We **must** extend the TCB to include SecDDR's security logic. This logic includes the secret key register, the encryption units for generating E-MACs, and the attestation logic.
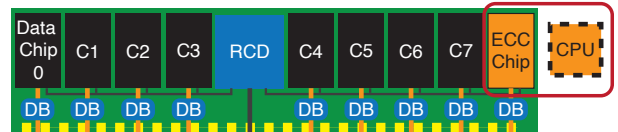

Fig. 5: SecDDR's hardware TCB. CPU and the ECC chip(s) are in the TCB. The ECC chip(s) contains the security logic.

Figure 5 shows SecDDR's TCB.[4] A powerful adversary can initiate the attack from within the DIMM by tapping or tampering with the on-DIMM components or using a malicious DIMM. To defeat these scenarios, we place SecDDR's security logic within the the ECC chip(s), making the ECC chip(s) part of the TCB. Only trusting the ECC chip(s) is sufficient to detect all active attacks, as tampering with the data chips will cause a MAC mismatch. If the memory module has multiple ranks, the ECC chip(s) in each rank are independent. The processor must establish a separate secure E-MAC channel and use a different transaction counter for each rank.

---

[3]We use half the DDR data rate for the CCCA signals [36].

[4]To fairly compare with prior work [2], we discuss SecDDR's compatibility with trusted memories in Section VI-C.

F. Initialization & Attestation in SecDDR

Memory Attestation. We adopt an attestation protocol similar to prior proposals [2], [6], [49]. The memory manufacturer embeds *endorsement* public-private keys (EK_p and EK_s) in each rank's ECC chip(s). While EK_p is accessible for attestation, EK_s never leaves the chip.

At each power up or DIMM replacement, the processor and each rank use public-key encryption to agree on a new shared key, i.e., the transaction key (K_t). We use an authenticated key-exchange protocol that is protected against impersonation and man-in-the-middle attacks [8], [9]. For authentication, we use EK_s to sign the memory module's key-exchange messages. The DIMM's certificate and EK_p should also be shared with the processor to authenticate the key-agreement and to be verified against the DIMM's certificate via a trusted *certificate authority (CA)*. This information can be shared either during the key exchange [9], or it can be manually entered by the *trusted* system-integrator [6]. The CA can be the memory vendor or a third party. Certificates may be cached in system-encrypted memory and periodically checked against revocation lists.

After confirming the memory module's identity and sharing K_t, the processor and memory agree on a common C_t. The processor chooses an initial counter value for each rank and transfers it to the DIMMs. C_t can be shared in plain-text and does not require integrity protection; tampering with the counter results in counter mismatch between the processor and memory, which will be detected through MAC verification failures. We can initialize the counter with a random number, or we can use a non-volatile register for the processor's C_t to use monotonically increasing values for the processor lifetime.

Finally, the processor actively clears memory (writing zeros) to protect from DIMM substitution or replaying stale pre-boot state. Note that attestation is infrequent (at each boot, DIMM power-up, or after non-adversarial DIMM replacement) and only incurs a slight slowdown at that time (seconds). DRAM-timing re-calibration, wake-up from sleep, and other cases where DRAM is already initialized do not require attestation.

Remote Attestation. SecDDR's DIMM attestation and establishing the secure E-MAC channel on the bus are transparent to the remote attestation of the processor and the software running in an enclave. We can use the same protocol as in SGX to attest the processor, retrieve an enclave's measurement, and create a secure channel to a remote client [8].

IV. EXPERIMENTAL METHODOLOGY

A. Simulation Framework

Simulator. We use Scarab [34] for simulation. Scarab uses Intel Pin [31] as the functional model. Main memory is modeled using Ramulator [25]. The virtual page size is 4KB with random policy for virtual page to physical frame mapping. Table I shows the configuration parameters.

Workloads. We use the SimPoints [50] methodology to create 200-million instruction representative regions of the SPEC-2017 [1] rate benchmarks and GAP Benchmark Suite

(GAPBS) [7]. Workloads with LLC MPKI ≥ 10.0 are considered as memory intensive. We simulate a 4-core system with each SimPoint replicated four times.

TABLE I: Configuration Parameters

Core	6-wide fetch/retire Out-of-order, 224 entry ROB, 97 entry RS, TAGE-SC-L branch predictor, 3.2GHz, 4 cores
L1 Cache	Private 32KB d- & 32KB i-cache, 64B line, 4-way
Last Level Cache	Shared 4MB, 64B line, 16-way
Prefetcher	Stream Prefetcher
Metadata Cache	Shared 128KB, 64B line, 8-way
Security Mechanisms	40 processor-cycles encryption and MAC
Main Memory	16GB DRAM, 1 channel, 2 ranks, 4 bank-groups, 16 banks, 8Gb_x8. 64 Read- and 64 Write-entry memory controller queues.
Memory Timings	DDR4-3200 at 1600MHz, tCL/tCCDS/tCCDL/ tCWL/tWTRS/tWTRL/tRP/tRCD/tRAS = 22/4/ 10/16/4/12/22/22/56 cycles

B. Evaluated Systems

We compare all configurations with a secure baseline that provides memory encryption and integrity protection, however, it lacks replay attack protection, to resemble Intel TDX as the state-of-the-art secure memory design in industrial products. Except in the encrypt-only configurations, MACs are placed in the ECC chips [10], [19]. We consider both counter-mode and AES-XTS encryption modes in our evaluation because they offer a security–performance tradeoff.

We compare 5 main system configurations. We also compare SecDDR directly to InvisiMem [2] in Section VI.

1) Baseline: The baseline secure system that follows recent academic work with a 64-ary integrity tree and counter-mode encryption. The integrity tree is built on the encryption counters with on-chip caching for both the tree and the encryption counters. We assume an idealized tree and encryption counters with no counter overflow. The encryption-counter lines and the tree nodes have the same number of counters (64). We allow parallel tree-level verification to reduce the overall verification latency. We do not allow speculative use of data.

2) SecDDR+CTR: SecDDR with the same counter-mode encryption as the baseline. We assume OTP_t latency can be hidden. For eWCRC, we increase the write burst length from 8 to 10 ($tBL = 5$ for writes). While eWCRC enhances the reliability for all configurations and has only a small performance impact, we make a conservative performance comparison and enable eWCRC *only* for SecDDR configurations. Note that for DDR5 memories the impact of increasing the write burst length is smaller – from 16 to 18 [36].

3) Encrypt-only, CTR: An upper-bound encryption-only secure system that assumes integrity rather than ensuring it.

4) SecDDR+XTS: A higher-performance variant of SecDDR that avoids counters overhead with AES-XTS encryption.

5) Encrypt-only, XTS: An AES-XTS encrypt-only system.

To model the AES-XTS encryption overhead, we add the encryption latency to each memory access. AES-XTS does not
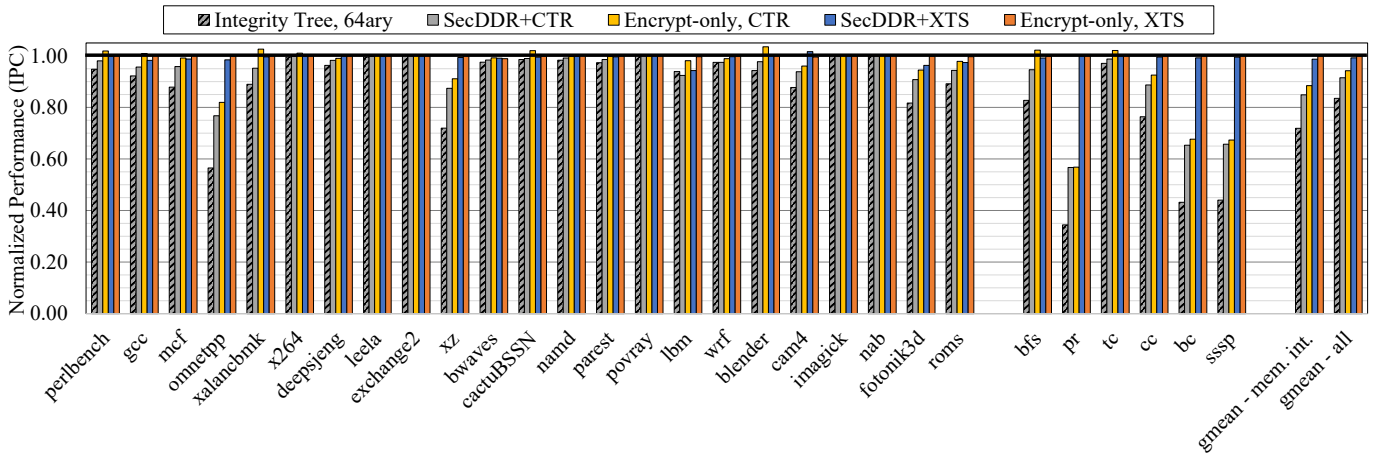
Fig. 6: Performance results. Normalized performance (IPC) to the Intel TDX baseline.

rely on encryption counters and encryption does not generate any additional memory requests. Although AES-XTS has been adopted as the industry standard for memory encryption [3], [18], its security guarantees are not identical to those of counter-mode encryption in SGX. Specifically, AES-XTS encryption no longer has *temporal* variation, meaning if a plain-text line (line-sized block at a specific memory address) has the same value at two different times, it will be encrypted to the same cipher-text, potentially leaking information [56]. Extending SecDDR as described in Section III to operate with counter-mode encryption is straightforward: Encryption counters are stored and cached as in the baseline secure system but their integrity is protected using per-line MACs, just like data.

V. EVALUATION

A. Performance Results

Overall Performance. Figure 6 shows the performance of the different configurations (total IPC) normalized to the TDX baseline. SecDDR+CTR improves average performance by 9.6% relative to the 64-ary tree baseline, performing within 3.0% of the encrypt-only system with counter-mode encryption. For memory intensive benchmarks, the average improvement is 18.0%. The largest speedup is observed for *pr*, *bc*, *sssp*, *omnetpp*, and *xz* that gain 64.7%, 51.2%, 49.4%, 35.9%, and 21.5%, respectively. These applications exhibit random memory access patterns and thus low locality. As a result, each data access requires traversing a different branch of the tree, which makes the metadata cache less effective, resulting in multiple off-chip accesses per data access for integrity tree traversal. Only *lbm* exhibits a small slowdown of 1.6% because it is write-intensive and is penalized by the longer write burst length of eWCRC that we only add to SecDDR.

Encryption Modes. Using AES-XTS, SecDDR+XTS provides 18.8% average performance improvement relative to the integrity-tree baseline and 5.4% better than counter-mode encrypt-only because it eliminates the overhead of accessing counters (37.7% and 11.6% for memory intensive workloads, respectively). SecDDR+XTS has negligible overhead (<1%)

compared with the encrypt-only system (with XTS), which is mainly caused by the extra write burst length due to eWCRC. *cam4* shows slight speedup (1.6%) with SecDDR+XTS, which appears to be from fewer prefetch requests due to better timeliness. However, this does not change the overall conclusion and we do not consider as an improvement of SecDDR.

A few benchmarks (*perlbench*, *gcc*, *xalancbmk*, *x264*, *cactuBSSN*, *blender*, *bfs*, and *tc*) exhibit higher performance with counter mode. These benchmarks have high locality and the counter cache has low enough miss rate (Figure 7) such that the latency saved by pre-computing *OTP* and hiding the encryption/decryption latency outweighs the overhead of fetching counters from memory. In contrast, AES-XTS never accesses counters but always incurs the encryption latency.
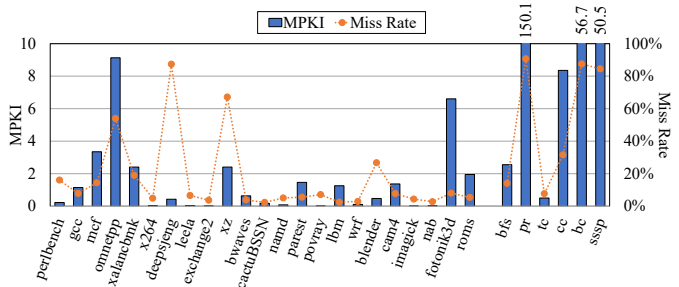

Fig. 7: Metadata cache behavior.

Note that in this comparison, the security guarantees of encryption modes in SecDDR+XTS and the integrity tree are not identical. However, assuming AES-XTS is acceptable for the security demands, applying state-of-the-art counter-based trees would be infeasible, and one needs to use hash-based Merkle Trees, which drastically hurts performance.

Sensitivity to Tree Arity. Figure 8 provides a comparison of different arity values to represent different tree types. The 128-ary design represents the state-of-the-art counter-based tree, MorphTree [45], which removes one level of the tree at the cost of greater complexity compared to our 64-ary baseline.

Compared with the 128-ary tree, SecDDR+CTR performs 6.3% better on average and has the advantage of scaling to high-capacity memories. Encrypt-only configurations with 64- and 128-counter exhibit similar performance. With 128-packing, each counter line spans two adjacent 4KB frames, but the random page mapping of our evaluation limits this advantage.
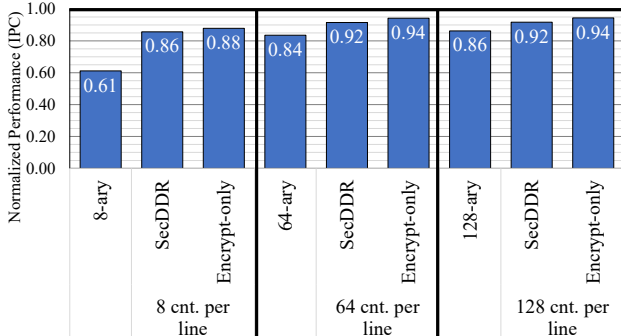

Fig. 8: Sensitivity to tree-arity and counter-packing.

We also evaluate the 8-ary design to represent hash-based Merkle Trees, which compute the hash over MACs rather than counters and can be used with AES-XTS. However, the performance penalty is very severe, incurring a 38.8% slowdown. Note that in addition to its compatibility with AES-XTS, the 8-ary design also differs in its placements of MACs. Instead of placing MACs in the ECC chips [10], [19], the 8 MACs that are hashed together in the tree should be placed in a contiguous block in memory. Otherwise, these MACs must be gathered from their respective locations, increasing the overhead. Thus, it is better to place ECC in the ECC chips in this design. The reliability of this organization is not identical to that of our other configurations, though prior work has established that the reliability is similar [10].

B. Area & Power Overheads

To estimate the area overhead of implementing SecDDR's security logic on the DRAM die, we follow the methodology of prior work [14], [58] that report numbers based on older generation of $45nm$ technology to account for the lower density and fewer metal layers in $\leq 22nm$ DRAM process technology.

E-MAC Generation. On the processor, we need a 16-byte register for the secret key (K_t), an 8-byte counter (C_t), and an AES unit. In each ECC chip, we need a 16-byte register for K_t, an 8-byte counter, and an AES unit. The AES engine can be implemented with $0.15mm^2$ area overhead using 45nm technology and provides encryption throughput of $53Gbps$ at $2.1GHz$ [33]. Table II summarizes the power overhead of AES engines in SecDDR. To estimate the power, we scale the power linearly assuming $500MHz$ DRAM core frequency and round the number of AES engines to meet the transfer rate of the chip, which is $12.8Gbps$ and $25.6Gbps$ for DDR4-3200 x4 and x8 chips, which results in the total of $70.8mW$ and $106.3mW$ extra power, respectively. SecDDR logic is only implemented in 2 out of 18 (x4) or 1 out of 9 (x8) DDR4 chips in each

rank (i.e., the ECC chips).[5] Considering 290-$350mW$ for one DRAM chip (9-$13W$ for a 16GB dual rank DDR4 DIMM) [38], the power overhead is less than 3% per-rank. For DDR5, an x4 DDR5-8800 chip requires encryption throughput of $35.2Gbps$, which results in the total of $89.3mW$ extra power for 3 AES engines operating at 1.1V. Assuming DDR5 memory consumes about 13% less power than DDR4 [47], the total overhead remains below 5%. Note that this is a conservative estimate given the $10nm$ *class* technology used in DDR5 [21].

TABLE II: AES engine power overhead (powers are in mW).

DIMM configurations and AES unit parameters	DDR4-3200, 1600MHz, 1.2V	
	x4 4Gb	x8 8Gb
AES units per ECC chip	2	3
AES power per ECC chip	70.8	106.3
DRAM chip power	290	351.9
16GB dual rank LRDIMM power	13230	9120
Overhead per rank	2.1%	2.3%

Thus, we expect the area overhead within the SecDDR device to be $< 1.5mm^2$, which is far less than the overhead reported for recent processing-in-memory (PIM) prototypes from memory vendors. These are capable of implementing much more complicated logic on the DRAM die [26], [29], [30], [40]. For instance, recent work [30] based on $20nm$ DRAM process technology reports $0.712mm^2$ area for a single PIM execution unit, which is more than $20\times$ larger than the AES engine after scaling from $45nm$.

Attestation. Key exchange and message signing require elliptic curve scalar multiplication and a hash function. Using $45nm$ technology, the multiplier can be implemented with $0.0209mm^2$ [32] and the hash function (e.g., SHA-256) with $0.0625mm^2$ [42] area overhead. At 1.1V and the peak operating frequency, these units consume $74mW$ and $50mW$ power at $3GHz$ and $1.4GHz$, respectively. Similar to the AES engine, scaling the power to $500MHz$, these units require 14.2 and $21mW$ extra power, respectively. Note that these units are only needed for attestation during system initialization, and can be turned off otherwise.

Multi-Channel Setting. SecDDR works for each memory controller independently, replicating the above logic per channel. We target untrusted DIMMs and each ECC chip in each rank implements the security logic.

VI. COMPARISON WITH INVISIMEM [2]

Using mutual authentication to protect the memory bus has been proposed by InvisiMem [2] for 3/2.5D-stacked memories. However, adapting InvisiMem to work with commodity DIMMs that operate using the DDRx protocol is impractical and requires fundamental changes to the memory system architecture. Although SecDDR set similar goals as InvisiMem, our approach is effective in addressing these challenges, making SecDDR suitable for adoption. In this section, we first provide a short background on InvisiMem and then, discuss these challenges.

---

[5]ECC is also encrypted to avoid leaking the plaintext MAC.

A. InvisiMem

InvisiMem[6] [2] uses the compute capability in 3/2.5D-stacked memories and the packetized protocol of the Hybrid Memory Cube (HMC) for address/command obfuscation and data confidentiality and integrity protection. InvisiMem extends the TCB to include the HMC logic layer (which contains the memory-side security logic) to create a mutually authenticated channel between the processor and HMC. InvisiMem takes advantage of the packet size flexibility [37] and adds new metadata to each HMC packet: a payload MAC that is used on the receiving end (processor on reads and memory on writes) to verify packet integrity and freshness.

InvisiMem deems physical attacks on the communication between the logic layer and DRAM cells impractical since in stacked memories, these connections go through the silicon layers (using through-silicon vias (TSV) or silicon interposer). This allows InvisiMem to effectively eliminate the integrity tree as the TSVs do not need replay-protection, and thus, storing a MAC with the data is sufficient to protect it while at rest (e.g., against Row-Hammer). On the memory side, after receiving a write request and verifying packet integrity, the security logic within the HMC generates a new MAC and stores it with the data to protect data at rest. On reads, the security logic first reads and verifies the stored MAC and then generates a new (channel) MAC for the packet using the transaction timestamp (C_t). Unlike InvisiMem, SecDDR avoids a mutually-authenticated channel and leaves all authentication to the processor, reducing complexity and latency in memory.

B. Challenges of Adapting InvisiMem for DDRx DIMMs

Adapting InvisiMem for commodity DDRx DIMMs introduces challenges that are not trivial to address.

Trusted HMC vs. Untrusted DIMM. The security-guarantees in InvisiMem are provided given that the logic layer in the HMC is trusted as part of the TCB and the 3D-stacked DRAM cannot be penetrated. This assumption, however, does not hold for commodity DDRx DIMMs. One could adapt InvisiMem for DDR DIMMs by placing the security logic in a discrete component on the DIMM (e.g., a buffer chip). However, this implementation does not protect the DIMM interconnects and other components on the DIMM, leaving the system vulnerable to malicious DIMMs and on-DIMM replay attacks. Thus, in addition to the security logic, the threat model must consider the **entire** DIMM as trusted (i.e., *trusted DIMM*).

To mitigate this problem, the security logic could be placed in the DRAM chips, however, this is infeasible because InvisiMem relies on memory-side integrity verification of every memory transaction; an operation that requires the entire packet payload (i.e., 64-Byte line) to be available. As opposed to an HMC in which all DRAM vaults are connected to a centralized logic layer, in DIMMs, data is distributed across multiple DRAM chips, which makes this approach impractical. Alternatively, the processor could create a separate secure channel to each DRAM chip, but this increases cost and requires increasing

the burst length to append MAC_t to each transaction (on both reads and writes).[7]

As a result, the approach proposed by InvisiMem is not suitable for commodity DDRx DIMMs that can contain malicious components and can be easily tampered with. On the other hand, even if the threat model with a trusted DIMM is an acceptable option, applying InvisiMem still requires fundamental changes in the design constraints of a modern DDRx DIMM, which we discuss next.

Packetized Protocol vs. DDRx. InvisiMem is particularly designed based on the packetized protocol in HMC, which has been deprecated [39]. However, DDR is not packetized, and given how widely it is accepted as an industry standard, changing this protocol is extremely difficult.

Additional Latency & Changes to the DDR Timing Parameters. InvisiMem's memory-side integrity verification adds additional latency to each memory access and requires changing the DDR timing parameters. Specifically, tCL must grow to account for the MAC latency. While this additional latency is deterministic (i.e., the MAC latency), it is on the critical path of every memory access.

Reduced Data-Rate due to the Centralized Buffer Chip. Implementing the memory-side verification and delegating the integrity-protection to the memory requires introducing a *centralized* data buffer on the memory module. That is, for memory-side MAC generation and verification, the memory module has to gather *all* 64 bytes of a line to compute its MAC. The centralized buffer, however, lowers the achievable memory frequency and bandwidth [41], [54].
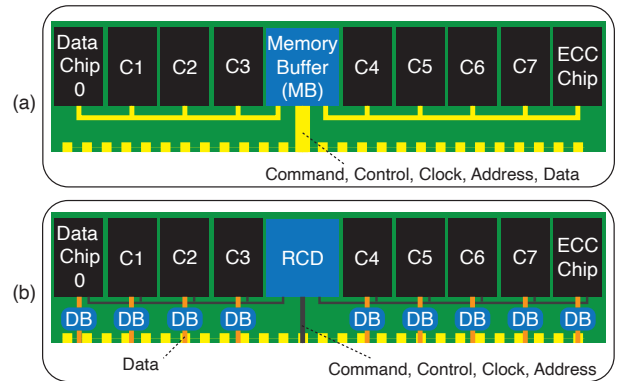

Fig. 9: Comparison between old (a) and new (b) DIMM architectures. (a) Centralized Memory Buffer in DDR3. (b) Centralized buffer for Control Signals and distributed Data Buffers in DDR4 and DDR5.

Figure 9a shows a DDR3 LRDIMM architecture with a centralized memory buffer chip (MB). All CCCA and data signals are first routed to the MB and then routed from the MB to each DRAM chip [5], [54]. The distance disparity (and thus,

---

[6]InvisiMem has two flavors. *InvisiMem_far* is related to this work.

[7]We can also protect the DIMM by implementing an integrity tree on the DIMM and placing its root in the buffer chip, however, this is not a scalable solution.
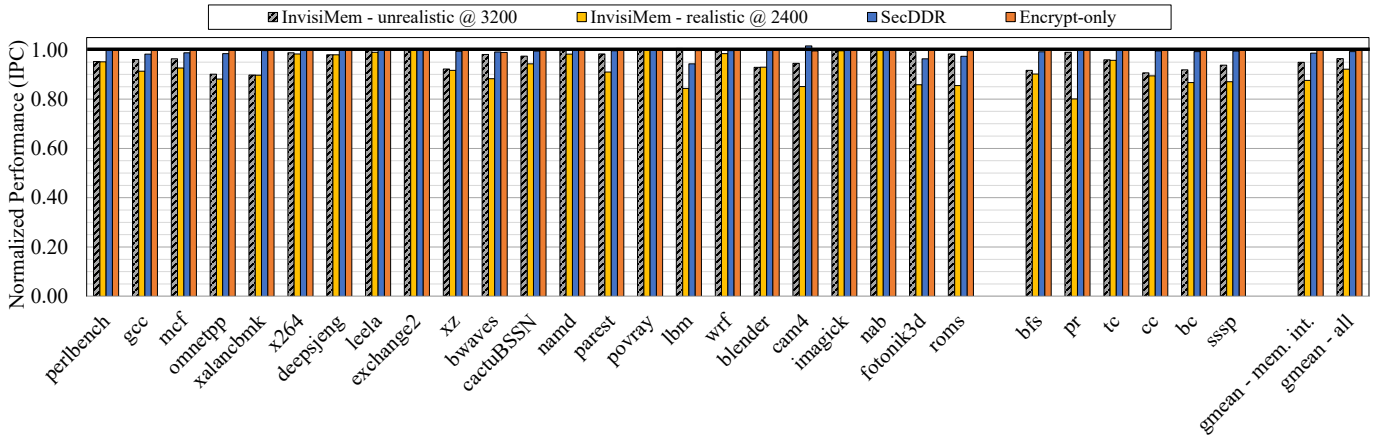
Fig. 10: Performance comparison with InvisiMem using realistic and unrealistic memory frequencies. Normalized performance (IPC) to the Intel TDX baseline. All configurations use AES-XTS encryption.

the latency) between different DRAM chips to the centralized MB limits the highest data rate and the frequency at which the DIMM can operate [5], [41], [54]. To address this problem, the buffer chips for the CCCA and data in DDR4 and DDR5 LRDIMMs are distributed [43], [54], as shown in Figure 9b. As discussed in Section II-A, The CCCA signals are buffered in the RCD chip, whereas the data is buffered in *distributed* DB chips. Distributed buffers are at a short and identical distance from their corresponding DRAM chips across the module, which reduces the buffering latency, enabling higher data rates. Adding a centralized buffer is undesirable and contradicts the main reason that newer memory technologies have transitioned to distributed data buffers.

C. SecDDR's Compatibility with Trusted Memory Modules

To provide an iso-secure baseline that mimics the case in which InvisiMem's [2] security logic is placed in a discrete component on the DIMM, we consider SecDDR with a *trusted* module that assumes on-DIMM attacks to be impossible, and discuss the placement of the on-DIMM security logic and the TCB components accordingly.
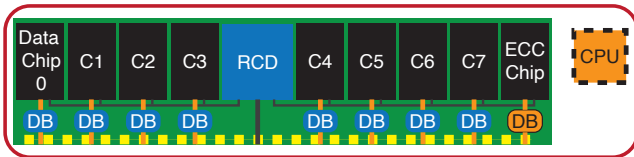

Fig. 11: SecDDR's TCB with a trusted DIMM. CPU and the DIMM are in the TCB. ECC DB contains the security logic and acts as the DIMM's root-of-trust.

As shown in Figure 11, we can apply SecDDR to a trusted module by placing the security logic inside the data buffer (DB) of the ECC chip(s). The ECC DB acts as the DIMM's root-of-trust, and is responsible for attestation and establishing the secure E-MAC channel. We assume the entire DIMM is in the TCB. Note that an attacker can perform a man-in-the-middle

replay attack by tampering with the DIMM interconnects or using a malicious DIMM that contains a hardware *trojan*. These attacks are impractical in the 3D-stacked HMC context of InvisiMem, but HMCs have proven non-viable in the market.

D. Performance Comparison with InvisiMem

SecDDR's E-MACs address all the above challenges. In contrast with memory-side MAC generation in InvisiMem, E-MACs are computed with no significant latency on the memory access critical path as the OTP_t can be pre-computed ahead of time. Furthermore, to encrypt MACs, counter-mode encryption only requires XOR-ing the MAC with the OTP_t that is performed cycle-by-cycle in the ECC DB or the ECC chip independently, and does not require any data communication or synchronization with the data chips. Thus, SecDDR does not require any centralized buffering, and it is compatible with the existing LRDIMMs that use distributed DBs.

Figure 10 shows the performance comparison between SecDDR and InvisiMem. We assume AES-XTS in all configurations. Only SecDDR is equipped with eWCRC and incurs the longer bursts necessary for it. To evaluate InvisiMem (with a *trusted* DIMM), we consider two cases: *unrealistic* and *realistic*. In the unrealistic case, we assume the memory can operate at 1600MHz (3200MT/s) and InvisiMem's overhead is only due to the $2\times$ MAC latency on the access critical path (one on the processor and one on the DIMM). Although this configuration has a small average overhead of 2.9% relative to SecDDR (3.8% on memory intensive applications), it is not achievable because the memory must run at a lower frequency due to the centralized data buffer. The realistic implementation operates at 1200MHz (2400MT/s) to account for this and InvisiMem then incurs a 7.2% average performance overhead relative to SecDDR (11.2% for the memory intensive applications).

Compared to the unrealistic idealized InvisiMem, SecDDR performs worse on *lbm*, *fotonik3d*, and *roms* by 6.6%, 3.0%, and 1%, respectively. This difference is due to the extra write burst length in SecDDR.
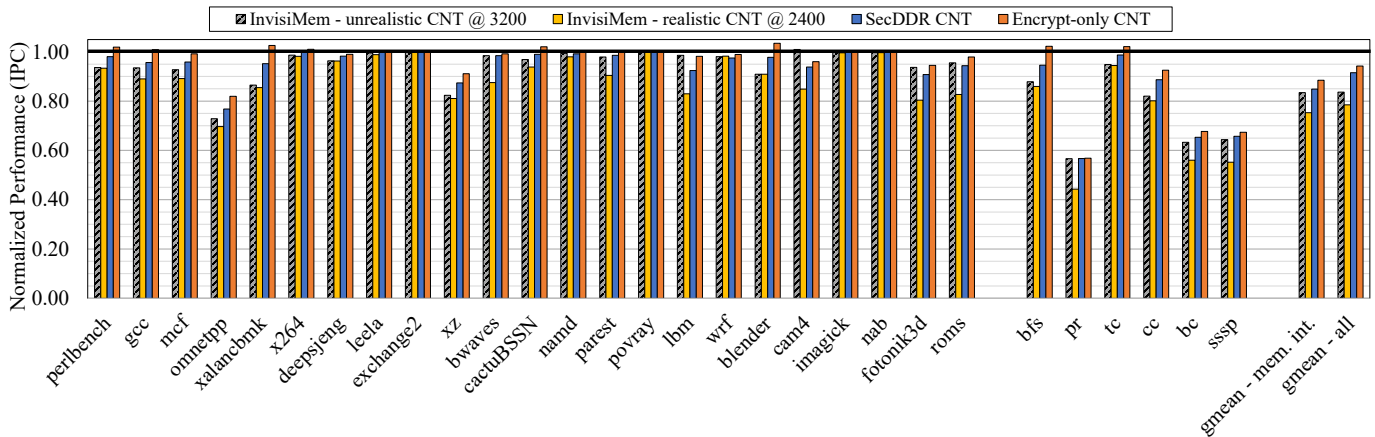
Fig. 12: Performance comparison with InvisiMem using realistic and unrealistic memory frequencies. Normalized performance (IPC) to the Intel TDX baseline. All configurations use counter-mode encryption with 64 counters per-line.

Figure 12 provides a similar comparison using counter-mode encryption. We observe a similar trend using counter-mode encryption and SecDDR outperforms InvisiMem unrealistic and realistic by 9.4% and 16.6%, respectively. Note, however, that AES-XTS is faster and provides higher overall performance (similar to Figure 6).

VII. OTHER RELATED WORK

Other Active Memory Designs. ObfusMem [6] was developed concurrently with InvisiMem and obfuscates the address and command buses with a point-to-point mechanism between the processor and the memory. Data integrity is delegated to integrity trees, limiting its scalability. SecureDIMM [49] provides address and command confidentiality by encrypting the memory bus, however, integrity protection is not provided. SecureDIMM uses Freecursive ORAM [11] to provide on-DIMM security. A similar proposal by Gundu et al. [16] off-loads integrity trees to the memory. Unfortunately, similar to InvisiMem, these designs rely on centralized buffer chips, which is not applicable to modern modules.

Reducing Overheads for Secure Memory. Synergy [46] is a reliability-security co-design that uses ECC to eliminate the MAC bandwidth overhead in ECC-DIMMs. SafeGuard [10] eliminates the storage and memory overhead of parities in Synergy. VAULT [53] builds a high arity tree by extending split-counters [57] to higher levels of a counter-based tree. Morphable-Counters [45] is a 128-ary tree that dynamically allocates more bits for frequently updated counters to reduce counter overflow. Taassori et al. [52] propose a compact tree design to reduce the parity update overheads in Synergy.

VIII. CONCLUSION & FUTURE WORK

Integrity trees are not scalable to protect large-scale memories due to the severe performance overhead. While mutual authentication is promising, existing proposals require fundamental changes in the memory system architecture. In this paper, we show that by identifying practical types of replay attacks, we can provide a low-cost scheme to protect the DDR interface against them. We propose SecDDR, which creates a reply-protected bus by encrypting the MACs. SecDDR has a negligible performance overhead and does not change the underlying DDR protocol. SecDDR can be extended to use the on-DIMM encryption units to encrypt the address and command for traffic obliviousness.

REFERENCES

[1] Spec 2017. https://www.spec.org/cpu2017.
[2] S. Aga and S. Narayanasamy, "Invisimem: Smart memory defenses for memory bus side channel," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 94–106. [Online]. Available: https://doi.org/10.1145/3079856.3080232
[3] AMD, "Amd memory encryption, white paper," Available Online.
[4] Apple, "Secure enclave, apple platform security," Available Online.
[5] H. Asghari-Moghaddam, Y. H. Son, J. H. Ahn, and N. S. Kim, "Chameleon: Versatile and practical near-dram acceleration architecture for large memory systems," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–13.
[6] A. Awad, Y. Wang, D. Shands, and Y. Solihin, "Obfusmem: A low-overhead access obfuscation for trusted memories," in *Proceedings of the 44th ISCA*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 107–119. [Online]. Available: https://doi.org/10.1145/3079856.3080230
[7] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015. [Online]. Available: https://github.com/sbeamer/gapbs
[8] V. Costan and S. Devadas, "Intel sgx explained," Cryptology ePrint Archive, Report 2016/086, 2016, https://ia.cr/2016/086.
[9] W. Diffie, P. C. Van Oorschot, and M. J. Wiener, "Authentication and authenticated key exchanges," *Designs, Codes and Cryptography*, vol. 2, no. 2, pp. 107–125, 1992. [Online]. Available: https://doi.org/10.1007/BF00124891

[10] A. Fakhrzadehgan, Y. N. Patt, P. J. Nair, and M. K. Qureshi, "Safeguard: Reducing the security risk from row-hammer via low-cost integrity protection," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 373–386.

[11] C. W. Fletcher, L. Ren, A. Kwon, M. van Dijk, and S. Devadas, "Freecursive oram: [nearly] free recursion and integrity verification for position-based oblivious ram," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 103–116. [Online]. Available: https://doi.org/10.1145/2694344.2694353

[12] B. Gassend, G. E. Suh, D. Clarke, M. Van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *HPCA-9 2003. Proceedings.* IEEE, 2003, pp. 295–306.

[13] Google. "half-double": Next-row-over assisted rowhammer. Available Online.

[14] P. Gu, X. Xie, Y. Ding, G. Chen, W. Zhang, D. Niu, and Y. Xie, "ipim: Programmable in-memory image processing accelerator using near-bank architecture," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 804–817.

[15] S. Gueron, "A memory encryption engine suitable for general purpose processors," Cryptology ePrint Archive, Report 2016/204, 2016, https://ia.cr/2016/204.

[16] A. Gundu, A. S. Ardestani, M. Shevgoor, and R. Balasubramanian, "A case for near data security," in *Workshop on Near-Data Processing*, 2014, Available Online.

[17] 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," *Commun. ACM*, vol. 52, no. 5, p. 91–98, may 2009. [Online]. Available: https://doi.org/10.1145/1506409.1506429

[18] Intel, "Intel® architecture memory encryption technologies specification," Available Online.

[19] Intel, "Intel® trust domain extensions (intel® tdx) module, base architecture specification," Available Online.

[20] Intel, "White paper, intel® trust domain extensions," Available Online.

[21] D. Kim, M. Park, S. Jang, J.-Y. Song, H. Chi, G. Choi, S. Choi, C. Kim, M. Han, K. Koo, Y. Kim, D. U. Lee, J. Lee, K. Kwon, B. Choi, H. Kim, S. Ku, J. Kim, S. Oh, D. Im, Y. Lee, M. Park, J. Choi, J. Chun, and K. Jin, "A 1.1-v 10-nm class 6.4-gb/s/pin 16-gb ddr5 sdram with a phase rotator-ilo dll, high-speed serdes, and dfe/ffe equalization scheme for rx/tx," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 167–177, 2020.

[22] J. S. Kim, M. Patel, A. G. Yağlıkçı, H. Hassan, R. Azizi, L. Orosa, and O. Mutlu, "Revisiting rowhammer: An experimental analysis of modern dram devices and mitigation techniques," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 638–651.

[23] J. Kim, M. Sullivan, S. Lym, and M. Erez, "All-inclusive ecc: Thorough end-to-end protection for reliable computer memory," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[24] Y. Kim, R. Daly, J. Kim, C. Fallin, J. H. Lee, D. Lee, C. Wilkerson, K. Lai, and O. Mutlu, "Flipping bits in memory without accessing them: An experimental study of dram disturbance errors," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. IEEE Press, 2014, p. 361–372.

[25] Y. Kim, W. Yang, and O. Mutlu, "Ramulator: A fast and extensible dram simulator," *IEEE Computer Architecture Letters*, vol. 15, no. 1, pp. 45–49, 2016.

[26] Y.-C. Kwon, S. H. Lee, J. Lee, S.-H. Kwon, J. M. Ryu, J.-P. Son, O. Seongil, H.-S. Yu, H. Lee, S. Y. Kim, Y. Cho, J. G. Kim, J. Choi, H.-S. Shin, J. Kim, B. Phuah, H. Kim, M. J. Song, A. Choi, D. Kim, S. Kim, E.-B. Kim, D. Wang, S. Kang, Y. Ro, S. Seo, J. Song, J. Youn, K. Sohn, and N. S. Kim, "25.4 a 20nm 6gb function-in-memory dram, based on hbm2 with a 1.2tflops programmable computing unit using bank-level parallelism, for machine learning applications," in *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 64, 2021, pp. 350–352.

[27] A. Kwong, D. Genkin, D. Gruss, and Y. Yarom, "Rambleed: Reading bits in memory without accessing them," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 695–711.

[28] D. Lee, D. Jung, I. T. Fang, C. che Tsai, and R. A. Popa, "An Off-Chip attack on hardware enclaves via the memory bus," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 487–504. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/lee-dayeol

[29] S. Lee, K. Kim, S. Oh, J. Park, G. Hong, D. Ka, K. Hwang, J. Park, K. Kang, J. Kim, J. Jeon, N. Kim, Y. Kwon, K. Vladimir, W. Shin, J. Won, M. Lee, H. Joo, H. Choi, J. Lee, D. Ko, Y. Jun, K. Cho, I. Kim, C. Song, C. Jeong, D. Kwon, J. Jang, I. Park, J. Chun, and J. Cho, "A 1ynm 1.25v 8gb, 16gb/s/pin gddr6-based accelerator-in-memory supporting 1tflops mac operation and various activation functions for deep-learning applications," in *2022 IEEE International Solid- State Circuits Conference (ISSCC)*, vol. 65, 2022, pp. 1–3.

[30] S. Lee, S.-h. Kang, J. Lee, H. Kim, E. Lee, S. Seo, H. Yoon, S. Lee, K. Lim, H. Shin, J. Kim, O. Seongil, A. Iyer, D. Wang, K. Sohn, and N. S. Kim, "Hardware architecture and software stack for pim based on commercial dram technology : Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 43–56.

[31] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 190–200. [Online]. Available: https://doi.org/10.1145/1065010.1065034

[32] S. Mathew, M. Kounavis, F. Sheikh, S. Hsu, A. Agarwal, H. Kaul, M. Anders, F. Berry, and R. Krishnamurthy, "3ghz, 74mw 2-level karatsuba 64b galois field multiplier for public-key encryption acceleration in 45nm cmos," in *2010 Proceedings of ESSCIRC*, 2010, pp. 198–201.

[33] S. K. Mathew, F. Sheikh, M. Kounavis, S. Gueron, A. Agarwal, S. K. Hsu, H. Kaul, M. A. Anders, and R. K. Krishnamurthy, "53 gbps native $gf(2^4)^2$ composite-field aes-encrypt/decrypt accelerator for content-protection in 45nm high-performance microprocessors," *IEEE Journal of Solid-State Circuits*, vol. 46, no. 4, pp. 767–776, 2011.

[34] HPS/SAFARI Research Group, "Scarab," Available Online.

[35] Micron. Ddr4 sdram features. Available Online.

[36] Micron. Ddr5 sdram, product core data sheet. Available Online.

[37] Micron, "Hybrid memory cube - hmc gen2," Available Online.

[38] Micron. Technical Note, Calculating Memory Power for DDR4 SDRAM. Available Online.

[39] Micron, "Micron announces shift in high-performance memory roadmap strategy," Available Online, 2018.

[40] J. Nider, C. Mustard, A. Zoltan, J. Ramsden, L. Liu, J. Grossbard, M. Dashti, R. Jodin, A. Ghiti, J. Chauzi, and A. Fedorova, "A case study of Processing-in-Memory in off-the-Shelf systems," in *2021 USENIX Annual Technical Conference (USENIX ATC 21)*. USENIX Association, Jul. 2021, pp. 117–130. [Online]. Available: https://www.usenix.org/conference/atc21/presentation/nider

[41] T. N. Platform. Optimizing ddr4 with server dimm chipsets. Available Online.

[42] R. Ramanarayanan, S. Mathew, F. Sheikh, S. Srinivasan, A. Agarwal, S. Hsu, H. Kaul, M. Anders, V. Erraguntla, and R. Krishnamurthy, "18gbps, 50mw reconfigurable multi-mode sha hashing accelerator in 45nm cmos," in *2010 Proceedings of ESSCIRC*, 2010, pp. 210–213.

[43] Rambus. Memory interface chips, ddr5 dimm chipset for servers. Available Online.

[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)*, 2007, pp. 183–196.

[45] G. Saileshwar, P. J. Nair, P. Ramrakhyani, W. Elsasser, J. A. Joao, and M. K. Qureshi, "Morphable counters: Enabling compact integrity trees for low-overhead secure memories," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2018, pp. 416–427.

[46] 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)*, 2018, pp. 454–465.

[47] samsung. DDR5: Twice as fast, with a 13% energy saving – and ready to power supercomputing and AI. Available Online.

[48] M. Seaborn and T. Dullien, "Exploiting the dram rowhammer bug to gain kernel privileges," *Black Hat*, vol. 15, p. 71, 2015.

[49] A. Shafiee, R. Balasubramonian, M. Tiwari, and F. Li, "Secure dimm: Moving oram primitives closer to memory," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 428–440.

[50] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS X. New York, NY, USA: Association for Computing Machinery, 2002, p. 45–57. [Online]. Available: https://doi.org/10.1145/605397.605403

[51] G. Suh, C. O'Donnell, I. Sachdev, and S. Devadas, "Design and implementation of the aegis single-chip secure processor using physical random functions," in *32nd International Symposium on Computer Architecture (ISCA'05)*, 2005, pp. 25–36.

[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)*, 2020, pp. 735–748.

[53] M. Taassori, A. Shafiee, and R. Balasubramonian, *VAULT: Reducing Paging Overheads in SGX with Efficient Integrity Verification Structures*. New York, NY, USA: Association for Computing Machinery, 2018, p. 665–678. [Online]. Available: https://doi.org/10.1145/3173162.3177155

[54] I. D. Technology. Ddr4 lrdimms, unprecedented memory bandwidth on samsung ddr4 lrdimm enabled by idt's register and data buffer. Available Online.

[55] A. Trikalinou and D. Lake, "Taking dma attacks to the next level," *BlackHat USA*, 2017.

[56] L. Wilke, J. Wichelmann, M. Morbitzer, and T. Eisenbarth, "Severity: No security without integrity : Breaking integrity-free memory encryption with minimal assumptions," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1483–1496.

[57] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," in *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, ser. ISCA '06. USA: IEEE Computer Society, 2006, p. 179–190. [Online]. Available: https://doi.org/10.1109/ISCA.2006.22

[58] A. Yazdanbakhsh, C. Song, J. Sacks, P. Lotfi-Kamran, H. Esmaeilzadeh, and N. S. Kim, "In-dram near-data approximate acceleration for gpus," in *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, ser. PACT '18. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3243176.3243188