ShieldCXL: A Practical Obliviousness Support with Sealed CXL Memory

KWANGHOON CHOI, School of Computing, KAIST, Daejeon, Korea (the Republic of)
IGJAE KIM, School of Computing, KAIST, Daejeon, Korea (the Republic of)
SUNHO LEE, School of Computing, KAIST, Daejeon, Korea (the Republic of)
JAEHYUK HUH, School of Computing, KAIST, Daejeon, Korea (the Republic of)

The CXL (Compute Express Link) technology is an emerging memory interface with high-level commands. Recent studies applied the CXL memory expanding technique to mitigate the capacity limitation of the conventional DDRx memory. Unlike the prior studies to use the CXL memory as the capacity expander, this study proposes to use the CXL-based memory as a secure main memory device, while removing the conventional memory. In the conventional DDRx memory, to provide confidentiality, integrity, replay protection, and obliviousness, costly mechanisms such as counter-based integrity trees and location shuffling by ORAM (Oblivious RAM) are used. Such mechanisms incur significant performance degradation in the current DDR-based memory systems, and their costs increase as the capacity of the memory increases. To mitigate the performance degradation, the prior work proposed an obfuscated channel for a secure memory module enclosing its controller in the package. Based on the approach, we propose a secure CXL-only memory architecture called *ShieldCXL*. It uses the channel encryption and integrity protection mechanism of the CXL interface to provide a practical ORAM while supporting confidentiality, integrity, and replay protection from physical attacks and rowhammers. To protect the PCIe-connected memory expanding board, this study proposes to use the standard physical sealing technique to detect physical intrusion. To mitigate the increased latency with the sealed CXL memory module, the study further optimizes performance by adopting an in-package DRAM cache. In addition, this study investigates destination obfuscation when a CXL switch is used to route among multiple hosts and memory devices. The evaluation shows that *ShieldCXL* provides 9.16x performance improvements over the prior ORAM technique.

CCS Concepts: • **Security and privacy → Hardware-based security protocols;**

Additional Key Words and Phrases: Hardware security, access obfuscation, CXL

ACM Reference Format:
Kwanghoon Choi, Igjae Kim, Sunho Lee, and Jaehyuk Huh. 2025. ShieldCXL: A Practical Obliviousness Support with Sealed CXL Memory. *ACM Trans. Arch. Code Optim.* 22, 1, Article 13 (March 2025), 25 pages. https://doi.org/10.1145/3703354

This work was supported by Institute of Information and communications Technology Planning and Evaluation (IITP) grants funded by the Ministry of Science and ICT, Korea (IITP2017-0-00466 SW StarLab and RS-2024-00396013) and National Research Foundation of Korea (RS-2024-00347114). This work was also partly supported by Samsung Electronics Co., Ltd. (IO201209-07864-01).
Authors' Contact Information: Kwanghoon Choi, School of Computing, KAIST, Daejeon, Korea (the Republic of); e-mail: khchoi@casys.kaist.ac.kr; Igjae Kim, School of Computing, KAIST, Daejeon, Korea (the Republic of); e-mail: ijkim@kaist.ac.kr; Sunho Lee, School of Computing, KAIST, Daejeon, Korea (the Republic of); e-mail: myshlee417@casys.kaist.ac.kr; Jaehyuk Huh, School of Computing, KAIST, Daejeon, Korea (the Republic of); e-mail: jhhuh@kaist.ac.kr.

ACM 1544-3973/2025/03-ART13
https://doi.org/10.1145/3703354

1 Introduction

To address ever-growing demands for memory capacity, the Compute Express Link (**CXL**) technology has emerged as a promising new memory-expanding technique with its abstract interface [8]. With its increased latency and lower bandwidth than conventional directly connected DDRx memory, the CXL memory has been proposed to act as a slow high-capacity NUMA node backing the fast main memory [43, 45, 51]. Several recent studies investigated the hardware architectures and OS supports for such CXL-based heterogeneous memory to mitigate the disadvantages of CXL memory by exploiting memory locality [36, 43, 45, 51].

Meanwhile, the protection of memory under software-based and physical attacks has become one of the critical requirements for systems processing security-sensitive data. For confidentiality and integrity support, hardware-based encryption and **message authentication code** (**MAC**) are used, and a counter-based integrity tree must be additionally maintained to protect memory blocks from replay attacks. Such an integrity tree has been known to reduce memory performance significantly as the memory capacity increases. Furthermore, as memory addresses also leak application behaviors, **oblivious RAM** (**ORAM**) hides memory access patterns by shuffling locations for every memory access. Although there have been significant improvements in ORAM techniques [50, 62, 63, 66, 75, 84], ORAM incurs 1.2x–5x execution time increases [75]. As such replay protection and ORAM supports do not scale well to future memory with multi-tera byte capacity, the memory protection technique must be revisited for the high-capacity memory.

A critical design constraint for the protection of DRAM is that it relies on the standard DDRx interface controlled by the in-CPU memory controllers. The memory controllers generate low-level commands to activate rows and access columns. The on-board paths to **Dual-Inline Memory Modules** (**DIMMs**) are vulnerably exposed to possible physical attacks. Unlike the conventional DDRx memory, a new CXL memory uses an abstract interface between the CPU and memory on top of the PCIe 5.0 interconnect and its memory modules are internally controlled by the in-device controller of the memory board. Such a new memory interface enables a totally different memory protection technique. Prior studies such as ObfusMem and InvisiMem proposed the memory protection based on encrypted command and data channels which hide both addresses and data by channel encryption [1, 2]. Based on the approach proposed by the prior work, we propose a practical solution for the emerging CXL technology, which has become a viable and widely accepted technology with its packetized memory interface.

As a novel solution for memory protection, this article proposes a CXL-only memory architecture, called *ShieldCXL*, which eliminates the conventional DDRx memory. As the vulnerable data paths to the conventional DIMMs are removed, the channels to the CXL memory device are encrypted and integrity-protected using the security support already existing in the CXL standard. As the command and address are also encrypted at flit granularity, the obliviousness (access obfuscation) is provided between the CPU and the controller of the CXL device. The design can eliminate the costly integrity tree and ORAM mechanism while it can provide all of the confidentiality, integrity, replay protection, and obliviousness supports. This study utilizes the fixed-sized configurable flit of CXL to efficiently hide differences in read and write accesses.

Although the channel between the CPU and CXL controller is protected by the encryption and MAC, the CXL controller eventually needs to access memory modules using conventional DRAMs. The CXL memory-expanding device is a discrete card connected to the PCIe interface. Therefore, it is possible to physically probe the exposed wires on the CXL memory device board. To address this vulnerability, we propose to use the standard physical sealing technology defined in **Federal Information Processing Standard** (**FIPS**) Publications 140-3. Such sealing techniques can detect any attempt to break in the sealed casing of the device and make the device unusable for any physical intrusion. Such techniques have been widely used for crypto-processors and other

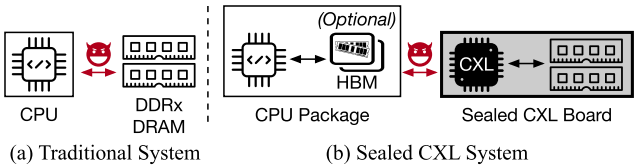


Fig. 1. The scope of physical attacks in a traditional DDR system versus a sealed CXL system.

security-critical devices [23–25, 54, 65, 71, 72]. Figure 1 presents the comparison between the traditional DDRx memory and the *ShieldCXL* memory. In Figure 1(a), the traditional system requires maintaining a counter tree and ORAM mechanism in the CPU with a very high cost. In Figure 1(b), the CXL memory device is sealed with physical intrusion protection. Only the channel between the CPU and CXL controller is protected by hardware-based encryption and MACs.

Based on the sealed CXL device with the encrypted channel, this study investigates the performance and scalability aspects of memory expansion. To mitigate the performance degradation by increased memory latencies of CXL devices, this study advocates to use an in-package DRAM cache integrated with the main CPU. Such an in-package DRAM can use LPDDR memory packaged together with CPU [17] or **High Bandwidth Memory** (**HBM**) [28], which are used in recent CPU designs. For scalable disaggregated memory architectures, we also explore the source-destination obfuscation with a CXL-based switch connecting multiple hosts and memory devices.

We evaluate the CXL-based memory protection techniques with a simulated system using ZSim and DRAMsim3 [46, 70]. For a set of memory-intensive applications, the CXL-only system with all security supports improves the performance by 9.16× on average, compared to the path ORAM with DDR4. Compared to the DDR-based secure memory without the ORAM support, which provides only confidentiality, integrity, and replay protection, *ShieldCXL* has an average performance improvement of 6.3% with a moderate HBM-based cache.

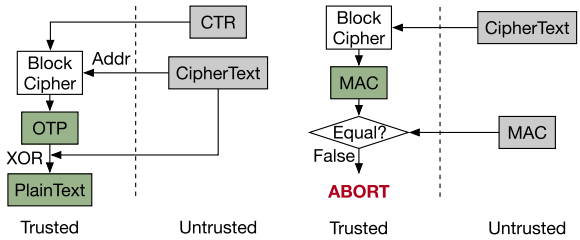
This study is the first study to use the standard CXL interface to provide obliviousness along with confidentiality, integrity, and replay protection. The main contributions are as follows:

— This article advocates to use the CXL memory as the main memory for security support. By strategically deploying physical tamper-responding sealing on a CXL device and implementing a proper CXL channel protection, it can solve the performance problem of the prior memory protection with an integrity tree and ORAM mechanism.
— By leveraging the fixed-size and configurable CXL flit, it proposes a flexible dummy scheme to efficiently hide memory request types. Additionally, it introduces an oblivious CXL switch which achieves access obfuscation utilizing the characteristics of the flit.
— Compared to ORAM-based access obfuscation technique, *ShieldCXL* achieves 9.16× performance improvement, where a small amount of in-package DRAM can further improve performance while maintaining security aspects.

2 Background

2.1 Hardware-based Memory Protection

To ensure data security, it is essential to guarantee confidentiality, integrity, and freshness. Confidentiality prevents data leakage by data encryption, while integrity ensures the detection of illegal data modification. However, integrity alone does not ensure that the data is the most recent version, necessitating additional mechanisms to ensure freshness. Moreover, the access pattern that contains addresses and commands can reveal program execution paths and privacy information, thus obliviousness (access obfuscation) should be provided.



(a) Counter-mode Encryption (b) MAC Authentication

Fig. 2. CTR-mode encryption and MAC authentication.

Counter-mode encryption: Figure 2(a) illustrates how a plaintext is derived from ciphertext using counter-mode encryption. The security engine utilizes a concatenated seed composed of a counter (CTR) and the address to generate **one-time pads** (**OTPs**) by block cipher. Ciphertext is transformed into plaintext through a 1-cycle XOR operation with the OTP, and vice versa. While the address used in OTP provides spatial uniqueness of data, the counter indicates the number of write-backs, providing temporal uniqueness.

MAC authentication: Tampering attacks involve unauthorized modification of data, leading to unexpected operations. To ensure integrity, tampering is detected using an MAC as shown in Figure 2(b). When data exits the secure boundary, the security engine generates a MAC using the data, which is then transferred along with the data. Before the data is processed, the MAC generated from the data is compared to the previously generated MAC to detect data modification.

Freshness verification: Although data encryption and MAC authentication are implemented, replay attacks are still feasible, where a stale data-MAC pair can bypass MAC verification. To ensure data freshness, a counter is used to represent the version of the data. It implies that if the freshness of the counter is guaranteed, then the freshness of the data is also assured. Recently, many studies have been focusing on the integrity tree to ensure counter freshness [10, 56, 67, 69, 78, 85].

Each data cacheline is assigned a counter which represents the version of the data. These counters form the leaf nodes of an integrity tree, and their parent nodes are recursively constructed up to the root by hashing their child nodes values. The root is always stored in a secure region and counter-verification is achieved through a hash comparison up to the root. To save the fetching of counters and tree nodes, the counter cache and tree node cache are utilized to store these tree nodes in a secure on-chip cache.

Replay protection for DDR interface without integrity tree: SecDDR leverages counter-**encrypted MACs** (**E-MACs**) to provide a low-cost protection mechanism for replay attacks on the DDR interface [12]. It utilizes synchronized counters between the CPU and the ECC chip, where MACs are stored, ensuring the detection of replay attacks. However, as low-level DDR protocol has to transmit plaintext of addresses and commands through on-DIMM paths, it not only prevents SecDDR from providing access obfuscation but also necessitates additional mechanisms to guarantee data freshness. A replay attack can occur when the write address of a write request is modified, causing the user to access stale data that was supposed to be updated. SecDDR integrates the address into the OTP generation for write commands to mitigate such attacks, thereby increasing the critical path.

ORAM: Attackers can utilize access address and request type to extract sensitive information such as security keys [20, 31, 50, 86]. As a protection mechanism, Path ORAM is commonly employed to obscure access patterns [75]. In Path ORAM, memory is organized as a binary tree, where each

node contains multiple data blocks. Each data block is randomly assigned to a tree path, and when memory access occurs, all blocks in the tree path are fetched. These fetched blocks are temporarily stored in a client-side buffer known as the stash, and they are assigned a new path and eventually evicted to memory. Path ORAM achieves obliviousness by shuffling data locations and obscuring the access patterns from attackers.

Path ORAM incurs severe performance degradation as a single request always traverses the tree path and fetches the data mapped to the tree path. Also, Path ORAM requires more on-chip storage for stash and position map as the amount of protected data increases, which leads to performance degradation [14]. Ren et al. and Phantom implement the Path ORAM using a simulation and FPGA hardware, presenting 20% to 400% of latency on SPEC traces and SQLite queries [50, 66, 75]. To reduce the overhead, several previous studies utilize the access characteristics. LAORAM and PrORAM merge multiple requests into a superblock, and PageORAM expands a candidate of stash eviction by utilizing a page-granular DRAM access pattern [62, 63, 84]. Such mechanisms reduce the chance of stash overflow, thus enhancing the efficiency of the stash.

Access obfuscation with memory including its controller layer: ObfusMem and InvisiMem achieve access obfuscation with minimal performance degradation by using a memory with its controller layer such as **Hybrid Memory Cube** (**HMC**) or **Non-volatile memory** (**NVM**), which emerged as effective alternatives of ORAM [1, 2]. HMC features 3D-stacked DRAM connected through **Through Silicon Vias** (**TSVs**), ensuring that the internal structure of the HMC device is not exposed to attackers. As the interconnect between the CPU and HMC is exposed to attackers, access obfuscation is achieved through counter-mode encryption of transmitted commands, addresses, and data. However, HMC which is built on 3D stack technology comes with expensive costs and a limited capacity of 2 GB, resulting in low scalability [61]. On the other hand, NVM with its controller, such as a **phase-change memory** (**PCM**)-based DIMM, can be used However, it requires that any interconnections between the controllers and memory chips/dies are not exposed, and there is no clear description of how the NVM controller and memory chips/dies are integrated within a single package [2]. In contrast, CXL memory is expected to be widely used due to its larger scalability and versatility, and we propose adopting standard physical sealing for CXL memory devices.

2.2 CXL

Memory disaggregation via CXL: As the demand for greater memory capacity in applications has grown, hardware vendors have introduced a new interconnect standard. The new standard, CXL, enhances memory capacity by supporting high-bandwidth, low-latency, and high-flexibility memory protocol with cache coherency. Due to the serial interface of PCIe, higher bandwidth per pin can be achieved compared to DDR interconnect, where 12x PCIe 5.0 can provide 48 GB/s for read and 48 GB/s for write. Furthermore, numerous studies have indicated that the access latency of the CXL memory is comparable to that of DDR-attached memory [18, 77]. Additionally, CXL exhibits high flexibility when used as disaggregated memory. Traditional RDMA-based memory disaggregation requires software intervention and memory copy operations for effective communication between the host and memory devices over network interfaces [6, 19, 68, 83]. However, the CXL memory protocol allows direct access from CPU to CXL-attached memory devices using load and store instructions, eliminating the need for software intervention. A recent study proposed to use only the CXL interface as the main memory interconnect, replacing the conventional DDRx memory. However, the study seeks to exploit the higher bandwidth per pin of serial CXL interconnects to provide extra memory bandwidth without consideration of security support [7].

Flit-based control flow: CXL introduces the concept of flits (Flow Control Units) as the fundamental units for data transmission. As specified by CXL protocol, communication is orchestrated

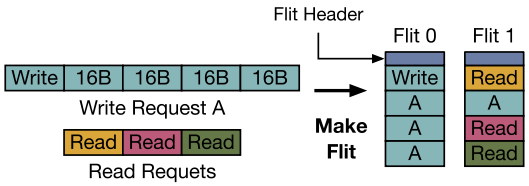


Fig. 3. CXL flit construction involving a write request and multiple read requests.

using flits that are either in 68 B or 256 B. We assume a flit size of 68 B in this work without loss of generality. As depicted in Figure 3, a 68 B CXL flit consists of a 4 B flit header, a 12 B header slot, and three 16 B generic slots. The CXL protocol defines slot formats for the header slot and generic slot, where each slot can hold memory requests/responses, data chunks, and MAC. The flit header encodes the slot format along with necessary transaction information such as flit type and acknowledgment.

This flit structure implies that multiple memory requests/responses and data can be encoded within a single flit. Flits are designed to flexibly configure slots for transmission efficiency, as illustrated in Figure 3, where a single flit can contain multiple memory requests and data. 64 B of data requires transmission across four 16 B generic slots, which may require spanning two flits. There also exists an all-data flit which consists solely of 64 B of data, utilizing all four slots and eliminating the flit header. Additionally, each flit maintains a fixed size, which distinguishes it from the variable-sized packet that carries a single request, which has been used in PCIe 5.0 or earlier. The configurable and fixed-sized flit provides a new opportunity to access obfuscation, which will be discussed further in the article.

CXL Integrity and data encryption (CXL IDE): The CXL IDE protocol outlines security measures to protect CXL transactions [8, 9]. AES-GCM is employed for flit encryption and authentication, which requires plaintext, **additional authentication data** (**AAD**), and an **initialization vector** (**IV**) for data encryption. The 4 B flit header included in the header slot serves as AAD. A counter included in IV monotonically increases for each flit encryption to prevent the reuse of IV. The entire flit without flit header is encrypted to ensure confidentiality. Besides, MAC generated with AES-GCM provides integrity for the flit. To enhance flit processing latency and bandwidth efficiency, CXL supports the aggregation of multiple flits to create a single MAC, known as communication MAC. The number of flits aggregated for communication MAC is referred to as a MAC epoch. Meanwhile, there are two configuration options for MAC verification, containment mode, and skid mode as in Figure 4. Containment Mode releases the flits only after the integrity verification passes. Aggregated flits arriving early must wait until the corresponding communication MAC is verified. On the other hand, in skid mode, flits are processed immediately upon arrival without waiting for MAC verification.

Challenges of supporting obliviousness on the existing TEE with CXL IDE: TEE combined with CXL IDE can support confidentiality and integrity but lacks support for access obfuscation. The CXL IDE provides flit encryption (excluding headers), MAC, and counter-mode encryption. Additionally, as the CXL IDE provides the channel protection only without the protection of DRAM, an integrity tree engine must be integrated into the CXL memory controller to ensure freshness for the CXL memory.

Several challenges arise when incorporating obliviousness. First, to guarantee the obliviousness of the CXL channel, it is necessary to encrypt the flit header which defines each slot type (read, write, data). Additionally, since an attacker can infer access types based on the number of flits transmitted, dummy flits must be generated. However, naively inserting dummy flits leads to

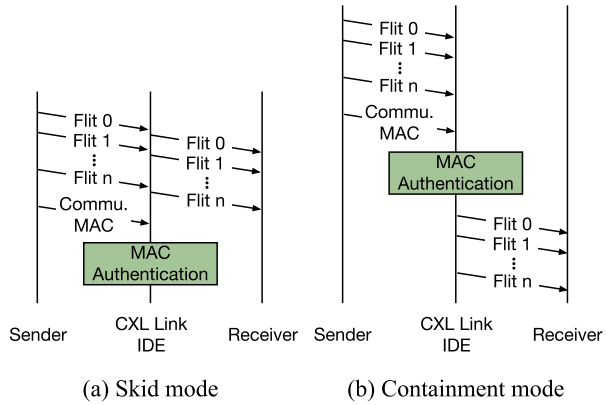


Fig. 4. Communication MAC authentication mode comparison: skid-mode vs. containment mode.

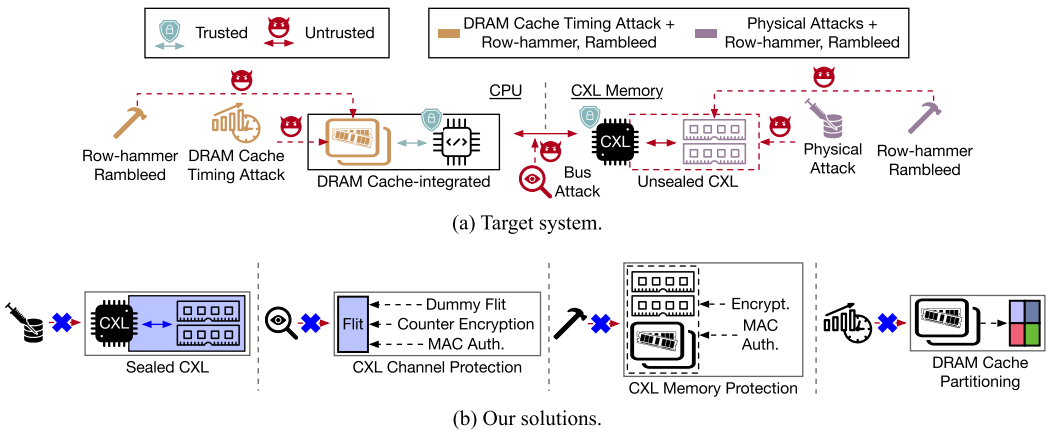


Fig. 5. Threat model. Available attacks are notated on a CPU package and an unsealed CXL memory device.

performance degradation. Second, to ensure the obliviousness of CXL memory where physical sealing is not applied, a Path ORAM controller must be integrated into the CXL controller. ORAM incurs larger performance overheads as memory capacity increases, due to the data shuffling required by its tree-based organization.

2.3 Threat Model

Figure 5 illustrates our proposed sealed CXL memory system architecture. The on-chip components of the CPU and CXL memory and its related software are considered as **trusted computing base** (**TCB**). CPU cores, the PCIe root complex, and in-package DRAM such as HBM are enclosed in the CPU package, thus physical attacks on these components are not feasible. CXL controller in a CXL memory device which translates CXL flits into DDR commands and vice versa, is also considered part of the TCB. In contrast, the interconnects between the CXL controller and memory modules within the CXL memory device are vulnerable to physical attacks unless they are protected by physical sealing, as depicted by the "Unsealed CXL" in Figure 5(a). However, "Sealed CXL" in Figure 5(b) demonstrates that with tamper-responding sealing, physical intrusion into the CXL memory device is detectable thus protecting itself from physical attacks.

The CXL channel, which lies on the PCIe physical lanes between the CPU and CXL memory, is vulnerable to physical attacks such as tampering and snooping. Attacks that do not require physical access like rowhammer and Rambleed are possible [37, 41], which can induce bit-flipping or read contents. Such attacks are feasible on in-package DRAM or memory modules within sealed CXL devices. The access control for security domains is provided by conventional trusted execution techniques such as Intel SGX [10], Intel TDX [29], and AMD SEV [33]. Such access control mechanism is orthogonal to memory protection against physical attacks.

Sharing resources inevitably opens side-channel. While adding a DRAM cache does not harm obliviousness, our threat model includes timing side-channel attacks targeting the DRAM cache [49, 82]. For instance, an attacker could attempt to infer sensitive data by exploiting variations in access times to the DRAM cache, leveraging a cache timing side-channel attack. However, we exclude conventional CPU cache timing side-channel attacks from our threat model as it is orthogonal to our memory protection and can be addressed by existing solutions [79]. Power, thermal, and electromagnetic side-channel attacks leaked by communication are not included in our threat model and can be addressed by existing solutions [3, 22, 39, 52, 53]. Furthermore, availability such as denial of service is not considered in this study.

3 Physical Tamper-Responding Sealing

3.1 Standard for Sealing Technology

The FIPS Publications 140-3 specifies the standard for security requirements for cryptographic modules, which is issued by the **National Institute of Standards and Technology** (**NIST**) [57, 58]. This standard establishes the security considerations satisfied by security modules, including authentication mechanisms, software/physical security, and mitigation of other attacks. FIPS 140-3 defines four security levels, and we will focus on the physical tampering-resistant feature provided in level 3 and above. Security level 3 is designed for detecting and responding to any attempts to physical access within a security module equipped with tamper-responding hard enclosures or circuitry. Upon detection of tampering, all **critical security parameters** (**CSP**), such as secret keys and authentication data, within the module will be zeroized. Additionally, security level 4 protects a security module against environmental failures such as power, voltage, and temperature hazards [58].

Tamper-response capabilities which are required by FIPS 140-3 level 3 can be achieved through the following sealing technology [4, 13, 30, 73]. The device features multiple layers of conductor grids known as sensing grids, which serve as evidence of tampering. The sensing grids are non-metallic, flexible, and embedded within a hard enclosure designed to visually and chemically resemble the enclosure, making it difficult for attackers to distinguish them. These grids are monitored by a circuit that detects any changes in the conductor properties. The entire package is enclosed in a grounded shield to reduce susceptibility to electromagnetic interference [30, 73]. If an attacker attempts to open the hard enclosure to access the internal module, the monitoring circuit detects changes in the conductor grid and triggers the tamper switch to indicate that tampering has occurred. Following the detection, the tamper switch activates the zeroization circuit, which erases CSPs and other sensitive data stored within the module.

Currently, numerous cryptographic modules employ these technologies to achieve FIPS 140-3 level 3 compliance [23–25, 54, 65, 71, 72]. Among them, the PCIe Cryptographic Security Module aligns with how the CXL memory device is physically constructed as a module [24]. It employs a tamper-responding enclosure on its PCIe module, thereby making penetration into the module unfeasible. Given the sealing technology combined into PCIe cryptographic modules, it is anticipated that seamless integration of these techniques with CXL memory modules is viable.

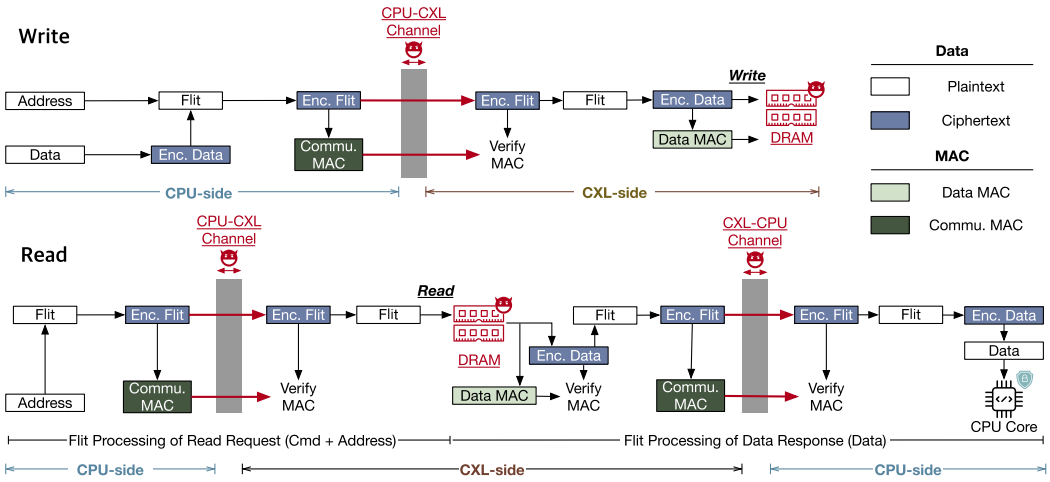


Fig. 6. Communication flows between CPU and CXL memory in *ShieldCXL*.

Ultimately, our goal is to leverage these measures to enable robust protection against a broader spectrum of attacks with low-performance overhead.

3.2 Challenges of Supporting Obfuscation by Sealing DDR DIMMs

One might consider applying tamper-responding sealing to DDR DIMMs to prevent on-DIMM paths from being exposed to attackers. However, while commercially available PCIe devices with tamper-responding sealing exist, no such implementation is currently available for DDR DIMMs.

Even if feasible, applying sealing to DDR DIMMs for access obfuscation poses challenges due to DIMM design. It is necessary to conceal the addresses and commands transmitted over the DDR memory bus. DIMMs contain a **registered clock driver** (**RCD**) that handles **command, control, clock, and address** (**CCCA**) signals [64]. However, encryption of commands and addresses at the RCD completely violates the current DDR standard and requires changing DDR timing parameters which are already optimized for current systems [12].

Given these challenges, applying tamper-responding sealing to DDR DIMMs is not feasible. It is beneficial to implement security-related functionalities in the hardware-abundant CXL controller, thereby providing confidentiality, integrity, freshness, and obliviousness with much lower complexity through sealed CXL memory.

4 Design

4.1 Overview

To eliminate the vulnerability through conventional DDRx interfaces, *ShieldCXL* does not use any external DRAM directly connected to the CPU. Instead, the CXL memory device is used as the main memory. The overall communication flows between CPU and CXL memory are shown in Figure 6. PCIe channel between the CPU and the CXL memory is protected by flit encryption and communication MAC. The channel encryption obfuscates memory addresses and commands supporting obliviousness in memory access patterns. The CXL memory device is sealed with the tamper-responding physical sealing with FIPS 140-3 level 3. Any physical intrusion will be detected, nullifying the channel secret key, which makes the stored data inaccessible. Note that the stored data are protected with confidentiality, integrity, and freshness by data encryption and MAC. The CPU package can contain the integrated HBM or DDR to use a secure in-package DRAM cache.

4.2 Device Authentication and Attestation

To ensure the security of the communication channel between the processor and CXL devices, it needs to establish device authentication between the secure processor and the CXL device. Device attestation assumes that device manufacturers, who are part of the TCB, have embedded asymmetric public and private key pairs in both the secure processors and memory. This assumption is realistic, as manufacturers like Intel already support such attestation between secure devices [10]. We design an attestation process compatible with **Secure Protocol and Data Model (SPDM)** [16], which is used by CXL IDE key management protocol. SPDM starts by exchanging versions and capabilities to ensure compatibility. The secure algorithm which is supported by both devices is selected to authenticate other devices. During authentication, the requester and responder exchange digests of manufacturer-signed certificates derived from their respective device public keys. These digests are signed with private keys to authenticate each party. The certificates are then verified by the trusted manufacturer who signed them, allowing for the secure exchange of public keys between endpoints. Once the public keys are exchanged, a symmetric session key can be securely transmitted using CXL_IDE_KM protocol.

4.3 Channel Protection Integrated with CXL IDE

Access obfuscation support with flit confidentiality: The CPU and CXL device communicate with each other using CXL flit where slot format is encoded in the flit header. In the existing design of CXL IDE, it only encrypts the flit slots, leaving the header exposed to attackers. Therefore, it lacks support for access obfuscation. To conceal not only the data and read/write addresses but also the request type, we encrypt both the slots and the flit header. This approach supports access obfuscation beyond what standard CXL IDE offers. The encryption process occurs at the link layer of the CPU and CXL memory, ensuring that the encrypted flit is securely transmitted over the vulnerable physical layer as in Figure 6. As provided in CXL IDE, it utilizes a synchronized counter pair embedded in both communication ends with AES-GCM. These counters are initialized with the same value and increment with each flit transmission. Monotonically increasing counter ensures that even identical data is encrypted differently with each transmission, which obscures temporal access pattern.

Flit integrity and freshness: In addition to information leakage through probing, data on transmission can be corrupted by active tampering. For example, attackers may directly manipulate the flit contents, and inject or drop flits on the channel. However, each flit is uniquely encrypted based on its counter, making it difficult for attackers to create meaningful flits for successful attacks.

To ensure flit integrity, we employ an MAC with a counter, as provided by CXL IDE. If an attacker modifies the content of a flit, the MAC generated by the modified flit will not match the previously generated MAC, leading to failure in the integrity check. Similarly, flit injection or flit drop will be detected by MAC which uses the synchronized counter. Besides, Attackers may attempt a replay attack by inserting previously used old flits into the CXL channel. However, the counter value used for encryption is different, resulting in attack detection. While using MAC for flit integrity verification may increase bandwidth usage, it can be mitigated by aggregating multiple flit MACs into a single communication MAC, as CXL IDE already supports it.

4.4 Request Type Obfuscation with Dummy Flits

While encrypting the entire flit prevents the leakage of request types and access addresses, it is still possible to infer request types by observing traffic from both directions. For example, a read request results in the transmission of a single flit, with the read request information embedded

in one slot. In contrast, a write request involves a write request information and four 16B data chunks, requiring a total of five slots, which translates to two flits.

Previous studies using variable-sized packets insert dummies to prevent attackers from inferring access types by observing traffic patterns [1, 2]. For example, the size of the packet that carries a single request can vary depending on whether the request is a read or write operation, allowing attackers to deduce the request type. To hide the request type, dummies are inserted into read packets to match the size of write packets so that attackers cannot distinguish read and write operations [1]. On the other hand, a read-then-write method can be used, where either a dummy or a real write packet always follows a read request [2]. However, these methods significantly increase bandwidth overhead, subsequently delaying memory request processing. Considering the bandwidth of a CXL channel using PCIe 5.0 x8 lanes is 32 GB/s per direction. Using a DDR5 DIMM which provides 32 GB/s or above could saturate the bandwidth. Introducing dummies would make it worse, leading to severe performance degradation due to a channel bandwidth bottleneck.

Leveraging the configurable flit which is of fixed size as in Figure 3, ensuring the same number of flits transmitted in both directions by inserting dummy flits is sufficient to prevent attackers from inferring the request type. Importantly, attackers cannot distinguish between real and dummy flits since they are counter-encrypted. Therefore, we can generate dummy flits only when the channel is idle, which alleviates the bandwidth overhead by the dummy and ensures the same number of transmitted flits at the same time. Note that dummy flits are generated and sent at flit-granularity, ensuring that CXL's capability to merge small requests into a single flit remains unaffected. A dummy slot is newly defined for CXL slot format and it is identified through the flit header, which is discarded upon decryption.

4.5 CXL Memory Device Protection

Data confidentiality: In this section, we focus on attacks targeting data stored in memory. While the tamper-responding sealing renders physical access to the CXL memory node unfeasible, it is necessary to protect data from attacks which do not require physical access, such as Rowhammer [37] and Rambleed [41].

Encryption can be executed in two possible locations: CXL controller in CXL memory or CPU. In our approach, we propose to encrypt data at CPU, as shown in Figure 6, for two reasons. First, we leverage the encryption engine already present in the CPU instead of adding hardware to the CXL-side CXL controller. We choose AES-XTS for data encryption, which is used for Intel **Total Memory Encryption** (**TME**). Second, it reduces the complexity when CXL memory is used with DRAM cache, which will be discussed later. By encrypting data at the CPU, it eliminates the need for redundant data encryption in two types of memory separately.

Data integrity: Data integrity should be ensured against data corruption attacks without physical access such as rowhammer attack. We propose to use MAC based on the SHA algorithm [59] for data integrity verification. With sealed CXL memory, we suggest data MAC verification on the CXL controller in CXL memory, as depicted in Figure 6, thus eliminating the bandwidth overhead by transmitting data MAC on the CXL channel. Data MAC is generated on a write operation, and stored in memory alongside the data. As the CXL memory node receives encrypted data from the CPU, MAC is generated on encrypted data. Then during a read operation, both the data and the MAC are retrieved by the CXL controller to verify if tampering has occurred.

Data freshness: Data freshness is ensured without the integrity tree owing to tampering-resistant sealing on the CXL memory module. Replay attack within CXL memory is impossible since attackers have no physical access to the CXL memory module. Replay attack via row-hammer and RAMBleed, which will be discussed later, are also not feasible.

4.6 Security Analysis

Protection on CXL channel and access obfuscation: Physical attacks on the channel are thwarted by flit counter encryption and communication MAC. Since the flit is encrypted, attackers cannot read the data, address, and request types within the flit. Flits of fixed size and an equal number of transmitted flits in each direction lead to access obfuscation. Any tampering attempts are detected by communication MAC verification. In the case of a replay attack, the mismatch of counter values in encryption indicates the detection of the attack.

Data security in CXL memory: Confidentiality attacks on CXL memory, such as RAMBleed and cold boot attacks, are mitigated since data are encrypted and then stored in memory. While rowhammer attacks can manipulate data stored in CXL memory, data MAC generated in the CXL controller guarantees data integrity. Previously generated MAC which was stored along with data is compared with the MAC of manipulated data to detect data tampering. A replay attack is not feasible in sealed CXL memory. RAMBleed needs access to the address of the content to carry out the attack. However, it is unable to reach the region where MACs are stored since an access control mechanism restricts unauthorized access. Even if the MAC is read, it is extremely challenging to change to the intended value with random position bit-flip by rowhammer. Alternatively, a cold boot attack can read the data-MAC pair, but the reboot process would change the key used for encryption, making the replay attack unfeasible.

Constant rate flit transmission: To mitigate the leakage of program execution paths through memory access time [15], we can adopt heart-beat flit transmission. Flits are transmitted to each direction at a constant rate, thus eliminating the leakage of memory access times and memory response times. It has minimal impact on performance since dummies are sent only when there is no real flit to transmit. However, as it may increase power consumption and ORAM does not address such an attack, we leave it as a design choice.

4.7 In-Package DRAM Cache

Recent processors widely adopt the integration of CPU and HBM or LPDDR [17, 28, 35, 76] to improve access latency and power consumption. From a security perspective, in-package DRAM is physically isolated from external access, which ensures that the data bus between CPU and DRAM is not exposed to attackers. Leveraging the fact that physical attacks are not feasible, we use the in-package DRAM as a secure cache to mitigate increased latency by CXL memory. However, as row-hammer attacks or Rambleed [37, 41] are still feasible in in-package DRAM, data encryption and MAC are employed in DRAM cache to mitigate these attacks.

There are two usages of in-package DRAM: (1) Extended main memory or (2) additional hardware cache. In-package memory can be used as extended main memory, where frequently accessed data is moved to in-package memory to utilize its lower access latency. On the other hand, in-package DRAM can be utilized as an additional hardware cache between the last level cache and main memory, and it shows better performance than when used as an extended main memory. Such a DRAM cache is commercially available and implemented as a hardware-managed, direct-mapped cache [26–28]. Cache conflict for direct-mapped cache should be considered carefully, and as described in [44], a minor modification to the operating system page allocation mechanisms can efficiently reduce cache conflicts. Besides, if an application fits into the capacity of HBM cache, allocating a fake NUMA node to the capacity of HBM cache can avoid unnecessary cache conflicts which occur due to physical memory fragmentation [27]. Therefore, as such DRAM cache designs are already implemented in commercial processors [26–28], we propose to leverage in-package DRAM as a DRAM cache, which improves performance and provides security features.

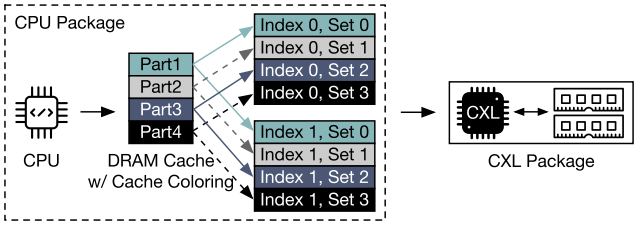


Fig. 7. The DRAM cache with partitions for security domains.

DRAM cache partition: We employ an in-package DRAM cache on top of *ShieldCXL* to support obfuscation and enhance performance at the same time. Adding a DRAM cache does not compromise the obliviousness provided by *ShieldCXL*, as its interconnect to the CPU is not exposed to attackers, and the CXL channel and CXL memory are protected as well. However, sharing resources like cache inherently introduces side channels, which can be exploited by attackers. Cache timing side-channel attacks leverage differences in access times to the cache, allowing inference of the victim's accessed address [47, 49, 82]. As we use DRAM cache with a commercially available direct-mapped cache design, isolation can be achieved by considering set partitioning only [38, 48, 81]. Not to add any extra hardware for partitioning, we use the traditional coloring to limit the cache space for each domain as in Figure 7. Each security domain (e.g., VM or enclave) is restricted to accessing only the cache sets assigned by the hypervisor or OS, achieved by mapping an ID to the set index. This approach effectively prevents cross-domain timing interference.

5 Obliviousness Support for CXL Switch

5.1 CXL Switch and Flit Routing

As the demands of data-intensive applications such as Large Language Models and big data analytics have been rising, it introduced significant challenges of memory capacity and bandwidth which become performance bottlenecks. With the advent of CXL 2.0, the CXL switch that connects CPUs, GPUs, accelerators, and memory is expected to offer a promising solution to such challenges. Through memory pooling enabled by CXL switches, it is possible to enhance memory capacity and bandwidth, overcoming the pervasive "memory wall" issue. Current research and development on CXL switches are being integrated into applications, indicating a trend toward their widespread adoption [18, 32, 34, 40, 80].

The primary role of a CXL switch is to manage traffic routing between CXL-enabled devices and the host processor. The CXL switch supports multiple **Virtual CXL Switches** (**VCS**), each equipped with its own upstream (CPU host) and downstream (CXL device) ports, enabling complex and flexible network topologies. Each CPU host connected to an upstream port in a VCS has its requests routed to the corresponding bound physical port, managed by the **Fabric Manager** (**FM**) via bind/unbind commands. Routing decisions, including access control, are determined based on the corresponding VCS and address range (or routing by port ID is available in CXL 3.0) with hardware components for efficiency. Address translation is executed by **Host-Managed Device Memory** (**HDM**) decoder, which is located either within the CXL switch or the CXL controller.

5.2 Access Obfuscation in CXL Switch

If the CXL switch is untrusted, numerous challenges arise in implementing access obfuscation. Since the CXL switch is unaware of routing, trusted communication channels must be established between the CPU and the CXL memory to set memory allocation information, and each CPU and CXL memory should maintain transaction counters and keys for every CPU-memory pair. Furthermore, as it is unclear which CPU and memory a memory request from the switch transfers to,

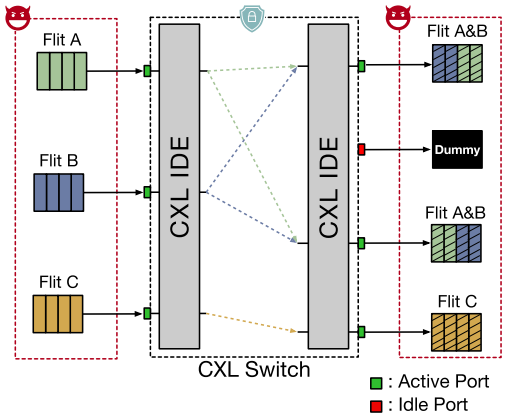


Fig. 8. CXL switch and access obfuscation support by flit re-encryption and dummy flits.

all memory requests need to be broadcasted from the switch to both CPU and memory. Each CXL memory and CPU has to verify every request to determine if it belongs to itself, and memory must send dummy responses to hide if it was actually accessed. This introduces significant hardware overhead and a substantial load on channel bandwidth.

Therefore, we propose an efficient obfuscation technique where only the routing component of the switch is trusted as shown in the Figure 8. Assuming that switch initialization and memory binding via the FM are securely conducted, having the routing component and CXL port within a physical package makes physical attacks infeasible. Trusting the routing component offers several advantages: Firstly, to hide which destination memory nodes the CPU is accessing, the switch sends dummies toward memory only but not to the CPU. Bandwidth can be significantly saved by utilizing our proposed flit dummy scheme. Dummy flits are transmitted in both directions only when the channel is idle to make the number of transferred flits the same. Secondly, flits that are re-encrypted before being transmitted from the switch appear completely different to attackers, thereby enhancing obliviousness. Additionally, it further complicates the access pattern due to the configurable flit, where flits from multiple nodes are reassembled into new flits before being sent to their destination node. Moreover, unlike previous methods, a switch capable of routing only needs to maintain a pair of a transaction counter and key per CXL channel directed towards either a CPU or memory. By trusting the routing component, we can achieve efficient and scalable obfuscation in the context of the CXL switch.

6 Evaluation

6.1 Methodology

Hardware configurations: We employ an execution-driven simulator, integrating ZSim [70] and DRAMSim3 [46], to model our proposed scheme. The simulation configuration details are presented in Table 1. We model an 8-core out-of-order processor with TLB and three levels of cache hierarchy. We use 2-channel DDR4 DRAM for DDR-attached and CXL memory, while the same DDR memory and HBM are employed for DRAM cache. CXL channel assumes an 8x lane PCIe 5.0 with a bandwidth of 32 GB/s per direction. Regarding CXL memory access latency, we account for the overall overhead of CXL protocol including transmission delay, re-timer, flit construction, and packet processing to reflect the actual access latency [5, 45, 77]. As in Table 1, access from CPU to CXL memory adds 80 ns of delay by the CXL port, while it takes an additional 180 ns when accessing via the CXL switch. The CXL switch is only used in the VI-E configuration.

Table 1. Hardware Configuration

CPU	Core	3.2 GHz, 8 cores, out-of-order x86_64
	L1 Cache	32 KB, private, 8-way
	L2 Cache	256 KB, private, 8-way
	L3 Cache	8 MB, shared, 16-way
CXL	Memory	DDR4-2400, 19.2 GB/s, 2 channels
	Bandwidth	PCIe 5.0, 32 GB/s per direction
	Latency	Port delay: 80 ns
Memory	DDR	DDR4-2400, 19.2 GB/s, 2 channels
	DDR Cache	DDR4-2400, 19.2 GB/s, 2 channels
	HBM Cache	HBM, 32 GB/s, 8 channels
AES Engine	870 MHz, 111.3 Gbps, 11 cycles per encryption	
CXL Switch	Switch and port delay: 180 ns (used in VI-E)	

We adopt a hardware-managed direct-mapped, 64 B block configuration for DRAM cache in line with existing commercial DRAM cache implementations [26–28, 44]. Cache conflict is minimized through page allocation policy as in [44]. Default size of DRAM cache is 256 MB, and we also evaluate various DRAM cache capacity which varies from 64 MB to 512 MB. Unsecure CXL with DRAM cache is evaluated to analyze the performance implications of DRAM cache on *ShieldCXL*. Experiments involving DRAM caches are under the assumption that they are partitioned by each security domain to mitigate timing side-channel attacks.

Security: For a realistic assumption, we employ a fully pipelined AES-GCM for counter mode encryption [11] and OTP buffers that can store up to eight OTPs per direction, as detailed in Table 1. We assume skid mode for flit authentication with flit MAC epoch set to 5. Data encryption in CPU utilizes AES-XTS (13 cycles), and SHA (40 cycles) is employed for data MAC in the CXL controller in CXL memory [42, 55, 85]. Our design includes four 16 KB, 8-way security metadata caches for counters, data MAC in CPU and CXL, and integrity tree nodes. For the counter tree scheme, we use a variable arity tree, VAULT [78] in DDR-attached memory (Secure DDR). We also evaluate the integrity tree-based TEE combined with CXL IDE (CXL VAULT), which can be considered a secure system baseline but lacks access obfuscation. Additionally, to compare *ShieldCXL* with a scheme supporting access obfuscation, we utilize an ideal Path ORAM in DDR-attached memory (DDR ORAM), which does not consider the cost of stash search and PosMap retrieval [75].

Benchmarks: We evaluate the performance of our scheme across diverse benchmarks from SPEC CPU 2006, SPEC CPU 2017, Biobench, **NAS Parallel Benchmark** (**NPB**), and Graph500. Table 2 illustrates **Last Level Cache misses per kilo instructions** (**LLC MPKI**) and **Instructions Per Cycle** (**IPC**) measured on unsecure DDR memory and memory footprints. Our experiments run eight copies of each benchmark, as well as a mixed application scenario classified by LLC MPKI where four benchmarks run with two copies each. We fast-forward the initialization phase and evaluate the results by simulating 1 billion instructions per core, except for three low-IPC workloads (mcf, graph500-list, and tiger) that run for 200 million instructions per core. In the mixed application scenario, as shown in Table 3, we configure 4 scenarios based on MPKI and adjust the instruction count of each benchmark to measure performance during simultaneous execution.

Our overall scheme and its security features are shown in Table 4. We present DDR and CXL-attached memory without security support as an unsecure baseline. Additionally, we evaluate the security overhead by implementing VAULT and Path ORAM on DDR-attached memory. Finally, we demonstrate the results of our scheme, *ShieldCXL* and with DRAM cache added, which shows improved performance while achieving the same security guarantee provided by Path ORAM.

Table 2. Benchmarks

	Workload (abbr.)	LLC MPKI	IPC	Memory footprint (MB)
Moderate MPKI (< 10.00)	namd (namd)	0.45	0.85	97.3
	ep (ep)	1.55	0.85	8.8
	ft (ft)	2.20	1.21	516.5
	gcc (gcc)	3.04	0.66	286.3
	cactusBSSN (bssn)	6.01	0.54	1,048.7
	cactusADM (adm)	6.54	0.34	1,196.4
	zeusmp (zeus)	8.99	0.55	1,723.9
High MPKI (≥ 10.00, < 40.00)	mg (mg)	15.99	0.40	3,915.6
	bt (bt)	19.37	0.36	5,510.5
	mummer (mum)	20.25	0.15	2,999.5
	xalancbmk (xal)	20.37	0.31	442.9
	graph500-csr (csr)	28.30	0.13	1,587.5
	sp (sp)	31.60	0.24	4,152.3
	graph500-list (list)	32.06	0.11	977.1
Extreme MPKI (≥ 40.00)	libquantum (quant)	42.47	0.24	256.4
	mcf (mcf)	75.88	0.08	2,035.2
	tiger (tiger)	300.67	0.02	4,383.3

Table 3. Applications for Mixed Scenarios

	Scenario ID	Workloads
Moderate: M High: H Extreme: E	MMMM	namd, ep, zeus, adm
	MHHH	adm, bt, xal, sp
	MHHE	namd, xal, sp, tiger
	MHEE	adm, xal, quant, mcf

Table 4. Scheme Overview

Scheme	Access Obfuscation	Confidentiality	Integrity	Freshness	Channel Protection
Unsecure DDR	✗	✗	✗	✗	N.A
Secure DDR	✗	✓	✓	✓	N.A
Unsecure CXL	✗	✗	✗	✗	✗
CXL VAULT	✗	✓	✓	✓	✓
DDR ORAM	✓	✓	✓	✓	N.A
ShieldCXL	✓	✓	✓	✓	✓

6.2 Performance Comparison with Path ORAM

To compare the performance of access obfuscation to *ShieldCXL*, we implement DDR ORAM as an ideal path ORAM executed in DDR-attached memory. Figure 9 illustrates the performance of *ShieldCXL* normalized to DDR ORAM. The benchmarks are ordered in ascending order of LLC MPKI. DDR ORAM is implemented as ideal where stash search cost is not considered and the entire PosMap is stored within ORAM controller in CPU. Nonetheless, DDR ORAM incurs significant overhead because it retrieves and fetches all data blocks along the tree path that contains the required data block. Furthermore, the more frequent the memory accesses, particularly as LLC MPKI increases, the more pronounced this trend becomes. *ShieldCXL* exhibits an average of 9.16× speedup compared to DDR ORAM, indicating *ShieldCXL* as a promising

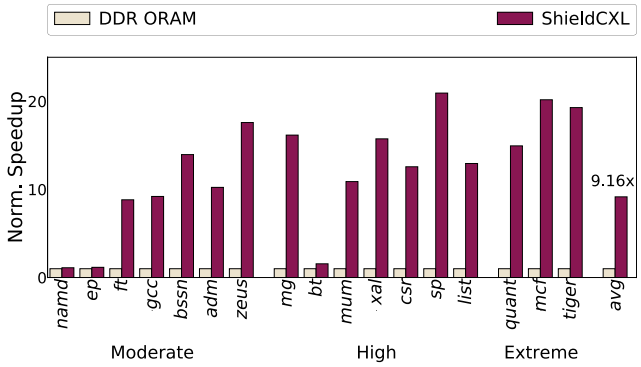


Fig. 9. Performance of *ShieldCXL* compared to an ideal Path ORAM. The performances are normalized to Path ORAM.

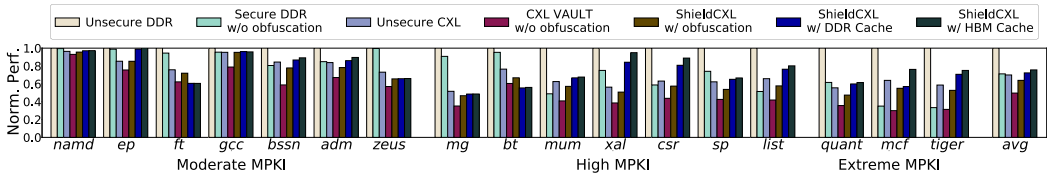


Fig. 10. Performance of *ShieldCXL* compared with others without obliviousness support. The performances are normalized to unsecure DDR and DRAM cache size is 256 MB. Note that only *ShieldCXL* provides access obfuscation.

solution for access obfuscation. It also demonstrates that *ShieldCXL* overcomes the limitations of DDR memory in terms of capacity and bandwidth while ensuring access obfuscation with improved performance at the same time.

6.3 Comparison of ShieldCXL to Schemes without Obliviousness Support

Single applications: Figure 10 illustrates the performance normalized to unsecure DDR-attached memory (unsecure DDR). Note that our scheme, *ShieldCXL*, provides all security features including access obfuscation while other schemes either ensure freshness or do not offer security measures.

CXL VAULT can be considered a secure baseline that provides confidentiality, integrity, and freshness, but it lacks access obfuscation. Additionally, it suffers performance degradation due to the need to fetch MACs, counters, and counter tree nodes simultaneously, resulting in a 28.7% performance drop compared to *ShieldCXL*. With high and extreme MPKI, both Secure DDR and CXL VAULT suffer even more significantly. It is prominent in benchmarks like mcf and tiger, characterized by a random access pattern compared to the streaming access pattern of quant. Specifically, *ShieldCXL* exhibits 1.24× and 1.60× speedup related to Secure DDR and CXL VAULT in Extreme MPKI applications, respectively. Moreover, *ShieldCXL* with DDR and HBM cache shows a speedup of 1.50× and 1.70× compared to Secure DDR, respectively. With moderate MPKI applications, the performance degradation of *ShieldCXL* with HBM cache (avg 84.1% of unsecure DDR) is minimized compared to Secure DDR without obliviousness support (avg 93.1% of unsecure DDR), even though CXL delay is included.

Mixed applications: As shown in Figure 11, we evaluate four mixed application scenarios (MMMM, MHHH, MHHE, MHEE) to examine where diverse workloads run in parallel. As the

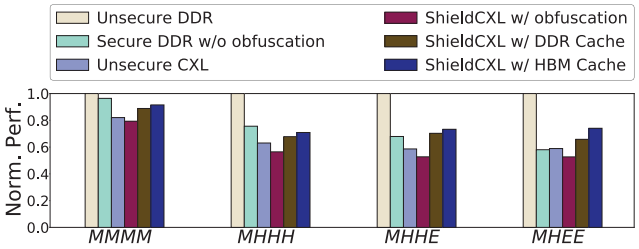


Fig. 11. Performance of mixed application scenario. The performances are normalized to unsecure DDR and DRAM cache size is 256 MB.

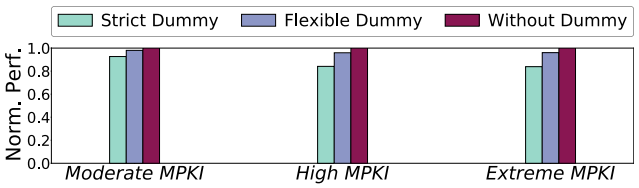


Fig. 12. Dummy scheme comparison on CXL memory. The performances are normalized to *ShieldCXL* without dummy.

number of memory-intensive workloads increases, performance degradation due to counter tree (Secure DDR without obliviousness support) becomes significant. *ShieldCXL* not only ensures access obfuscation but also exhibits competitive performance even in the presence of CXL delay, showing the effectiveness of our scheme across various scenarios.

6.4 Dummy Insertion with Configurable and Fixed-sized Flit

Dummy insertion is necessary to prevent inferring request types from traffic transmitted over the CXL channel. We applied the dummy scheme introduced by Invisimem to CXL flits, referred to as Strict Dummy. Strict Dummy generates dummies for each read request to make the number of flits the same as that of write requests [1]. Figure 12 compares the performance of Strict Dummy with our configurable flit-based dummy scheme, Flexible Dummy. Strict Dummy leads to notable performance degradation up to 18.6% compared to *ShieldCXL* without dummy due to the high burden on channel bandwidth. Additionally, pre-generated OTPs stored in the OTP buffer are consumed by dummy data first, which can lead to bottlenecks in generating OTPs for real requests.

In contrast, our proposed dummy scheme minimizes the impact on application performance by inserting dummy flits only when the channel is idle. Flexible Dummy reduces bandwidth usage and ensures that the OTPs are preserved for real requests, thereby enhancing overall system performance. As illustrated in Figure 12, our dummy scheme performs 98.0%, 95.9%, and 96.0% in each classified MPKI scenario to *ShieldCXL* without a dummy, indicating that the impact of the dummy on performance is minimized.

On the other hand, Obfusmem injects dummies to make every request appear as a read-then-write sequence to attackers [2]. This approach also can lead to significant bandwidth consumption where read requests are dominant, and write operations may be delayed as they await for read requests.

6.5 CXL Switch

In a CXL switch scenario, we assume a CXL switch is connected to four computing nodes and two CXL memory nodes. CPU accesses CXL memory through the switch which acts in the

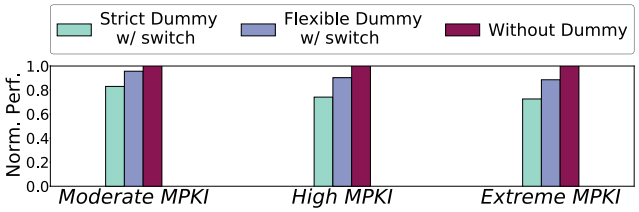


Fig. 13. Dummy scheme comparison on CXL Switch. The performances are normalized to *ShieldCXL* Switch without dummy.

middle, requiring a 2-level communication. As a result, dummy flits are generated on two separate channels for a single flit with a switch in between, meaning that the impact on bandwidth occurs at two points. Furthermore, to conceal which CXL memory node is being accessed, the switch must transmit both the actual request and dummy data to all CXL memory nodes upon receiving a request from the CPU.

Figure 13 demonstrates the effectiveness of our dummy scheme in the presence of a CXL switch. Using Strict Dummy in CXL switch exhibits a 22.6% slowdown compared to *ShieldCXL* without dummy due to the additional dummy traffic which increases with the number of CXL channels. However, utilizing the configurable and fixed-sized flit, Flexible Dummy minimizes the impact of dummies by transmitting them only when the channel is idle. As depicted in Figure 13, our dummy scheme achieves 92.2% of the performance of *ShieldCXL* switch without a dummy, thereby demonstrating its scalability and effectiveness even when a CXL switch is used.

6.6 Authentication Mode Choice

As in the CXL standard, where authentication modes such as skid and containment mode are supported, system designers can take these into account for their system designs to balance performance and security. Here we can consider a performance-enhancing approach, *All Lazy*, which extends the concept of skid mode to both CXL flit and data. On the other hand, where security is the priority, *Strict* scheme processes flits or data only after completion of each authentication process. However, *Strict* scheme incurs significant overhead. For example, when the CPU accesses CXL memory, three integrity verifications will be conducted serially: flit transmitted to CXL memory, data integrity checks at the CXL controller, and flit transmitted to CPU. To ensure both security level and performance, we propose *Strict CPU*, where all integrity checks are executed in parallel, but data processing in the CPU begins only after all checks have been completed. Data processing is initiated when the response from the latest of the three integrity checks reaches the CPU, strictly preventing any corrupted data from being processed within the CPU.

Figure 14 presents an evaluation of the three authentication modes. Strict scheme serializes the authentication process and data/flit processing, which incurs the most significant overhead due to the three serial authentication steps as described earlier. However, Strict CPU scheme, where authenticated data processing is guaranteed at the CPU, allows for the overlap of these three authentication processes. The bottleneck in Strict CPU mainly stems from the flit authentication on the CPU receiving the flit, which starts the latest among three integrity checks. Strict CPU achieves an average performance of 92.5%, 94.7%, and 95.5% compared to All Lazy with *ShieldCXL*, DDR cache, and HBM cache, respectively. This makes it a viable option for users who prioritize both performance and security.

6.7 DRAM Cache

Sensitivity to DRAM cache capacity: Figure 15 illustrates the performance of DRAM cache with capacity from 64 MB to 512 MB, normalized to the performance of 256 MB capacity for each

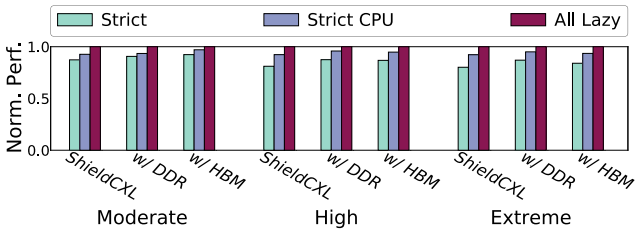


Fig. 14. Authentication mode comparison on *ShieldCXL*. The performances are normalized to All Lazy.

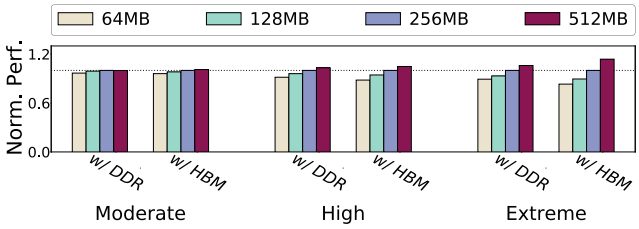


Fig. 15. Performance with various DDR cache capacity on *ShieldCXL*. The performances are normalized to 256 MB memory each.

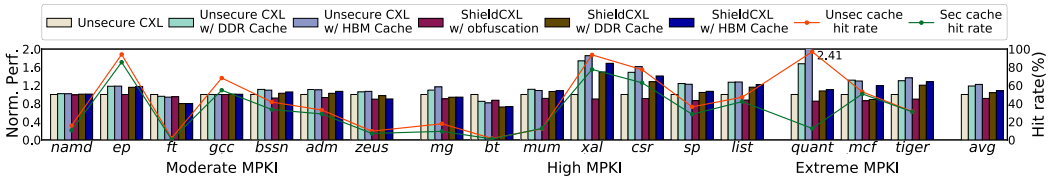


Fig. 16. 256 MB DRAM cache with unsecure CXL and *ShieldCXL*. The performances are normalized to unsecure CXL without DRAM cache. DRAM cache hit rates of 256 MB DDR cache on unsecure CXL and *ShieldCXL* are shown with the lines.

scenario. For low MPKI benchmarks, low number of LLC results in less noticeable performance differences with varying capacities. In high MPKI scenarios, increasing the capacity leads to performance improvement due to its memory-bound behavior. Using HBM as DRAM cache, it shows distinct performance improvement as its capacity grows. Considering that current processors [28] support HBM as DRAM cache, performance degradation by CXL memory can be mitigated with HBM cache, and DDR can be considered a viable option as well.

Comparison to unsecure DRAM cache: Figure 16 illustrates the performances of DRAM cache with unsecure CXL and *ShieldCXL*. To demonstrate the effect of DRAM cache, it also presents the 256 MB DDR cache hit rates for both unsecure CXL (Unsec cache) and *ShieldCXL* (Sec cache) scenarios. Unsecure CXL with a DRAM cache does not have CXL IDE enabled, thus no protection for CXL channel. Additionally, there is no data protection, such as encryption or MAC, and the DRAM cache partitioning, which mitigates timing-based side-channel attacks, is not enabled. Due to these factors, as shown in Figure 16, unsecure CXL with DRAM cache shows a performance increase of 14.3% in DDR and 13.0% in HBM compared to *ShieldCXL* with DRAM cache. Additionally, the DRAM cache hit rate in *ShieldCXL* decreases due to partitioning, as each security domain can only use its allocated portion of the DRAM cache, leading to more cache conflicts. The significant difference in DRAM cache hit rate and performance for quant is due to its sequential access pattern, causing frequent evictions before data can be reused due to the smaller allocated cache space.

Nevertheless, using in-package DRAM cache with consideration for cache conflicts can enhance performance while maintaining security [27, 44].

7 Discussion

Comparison of ShieldCXL to prior work: Our work builds on the general approaches proposed by the previous two studies [1, 2]. A key innovation is how we adapt these ideas to the new CXL technology, which has become a viable and widely accepted standard with packetized memory interfaces. We focus on identifying the additional mechanisms and policies required to efficiently incorporate all three aspects of memory security—confidentiality, integrity, and obfuscation—into CXL-based memory systems.

ShieldCXL utilizes the unique characteristics of CXL to offer practical and efficient solutions in its context. Firstly, we minimize bandwidth overhead caused by dummies by leveraging the fixed size of CXL flits. We discover that since CXL flit is fixed-size, attackers could infer the access type based on the number of flits transmitted. Therefore, it only needs to match the number of flits transmitted in both directions, which minimizes the bandwidth overhead by inserting dummies only when the channel is idle. Secondly, this article proposes an access obfuscation technique specifically for a CXL switch. Achieving access obfuscation in a switch that connects multiple hosts and memory devices is highly complicated. However, by leveraging the characteristics of CXL flits, our dummy scheme efficiently conceals the destination of transactions, as illustrated in Figure 13. Thirdly, *ShieldCXL* is compatible with CXL IDE. By extending the existing CXL IDE, minor hardware modifications such as an encryption engine for encrypting entire flits and a component for dispatching dummies are required. Lastly, this article explores and optimizes the authentication modes of CXL IDE as discussed in Section 6.6.

Since the work is based on CXL technology, it can be enhanced by leveraging the expandability and versatility of CXL. CXL memory can use various memory technologies such as DDR, LPDDR, and HBM, offering flexibility to meet different requirements. Additionally, although this research focuses on CXL memory, the approach can also be applied to CXL devices in PCIe card form factors, such as CXL Smart NIC [21, 74], CXL PNM [60], and CXL accelerators. This makes the proposed technique broadly applicable to systems where access obfuscation is needed.

8 Conclusion

This article proposed a practical ORAM with a sealed CXL device, which provides confidentiality, integrity, replay protection, and obliviousness under physical attacks and rowhammers. Inspired by the prior techniques with the memory including its controller layer, it greatly simplifies the memory protection eliminating the costly mechanisms such as counter-based integrity tree and memory block shuffling for ORAM. With the CXL design, it provides a practical solution for memory protection with high memory capacity.

References

[1] Shaizeen Aga and Satish Narayanasamy. 2017. InvisiMem: Smart memory defenses for memory bus side channel. In *Proceedings of the International Symposium on Computer Architecture (ISCA'17)*. 94–106.

[2] Amro Awad, Yipeng Wang, Deborah Shands, and Yan Solihin. 2017. Obfusmem: A low-overhead access obfuscation for trusted memories. In *Proceedings of the International Symposium on Computer Architecture (ISCA'17)*. 107–119.

[3] Shivam Bhasin, Paolo Maistri, and Francesco Regazzoni. 2014. Malicious wave: A survey on actively tampering using electromagnetic glitch. In *Proceedings of the International Symposium on Electromagnetic Compatibility*. 318–321.

[4] J. A. Busby, E. N. Cohen, E. A. Dames, J. Doherty, S. Dragone, D. Evans, M. J. Fisher, N. Hadzic, C. Hagleitner, A. J. Higby, M. D. Hocker, L. S. Jagich, M. J. Jordan, R. Kisley, K. D. Lamb, M. D. Marik, J. Mayfield, T. E. Morris, T. D. Needham, W. Santiago-Fernandez, V. Urban, T. Visegrady, and K. Werner. 2020. The IBM 4769 cryptographic coprocessor. *IBM Journal of Research and Development* 64, 5/6 (2020), 3:1–3:11.

[5] Businesswire. 2021. PLDA and AnalogX Announce Market-leading CXL 2.0 Solution featuring Ultra-low Latency and Power. Retrieved 11 November 2024 from https://www.businesswire.com/news/home/20210602005484/en/PLDA-and-AnalogX-Announce-Market-leading-CXL-2.0-Solution-featuring-Ultra-low-Latency-and-Power

[6] Irina Calciu, M. Talha Imran, Ivan Puddu, Sanidhya Kashyap, Hasan Al Maruf, Onur Mutlu, and Aasheesh Kolli. 2021. Rethinking software runtimes for disaggregated memory. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*. 79–92.

[7] Albert Cho, Anish Saxena, Moinuddin Qureshi, and Alexandros Daglis. 2024. COAXIAL: A CXL-centric memory system for scalable servers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC'24)*.

[8] CXL Consortium. 2024. CXL 3.1 Specification. Retrieved 11 November 2024 from https://computeexpresslink.org/cxl-specification/

[9] CXL Consortium. 2024. Integrity and Data Encryption (IDE) Trends and Verification Challenges in CXL. Retrieved 11 November 2024 from https://computeexpresslink.org/blog/integrity-and-data-encryption-ide-trends-and-verification-challenges-in-cxl-compute-express-link-2797/

[10] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. In *Proceedings of the Cryptology ePrint Archive*.

[11] Pham-Khoi Dong, Hung K. Nguyen, and Xuan-Tu Tran. 2019. A 45nm high-throughput and low latency AES encryption for real-time applications. In *Proceedings of the International Symposium on Communications and Information Technologies (ISCIT'19)*. 196–200.

[12] A. Fakhrzadehgan, P. Ramrakhyani, M. K. Qureshi, and M. Erez. 2023. SecDDR: Enabling low-cost secure memories by protecting the DDR interface. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'23)*. 14–27.

[13] Elischa Ferres, Vincent Immler, Alexander Utz, Alexander Stanitzki, Reneé Lerch, and Rainer Kokozinski. 2018. Capacitive multi-channel security sensor IC for tamper-resistant enclosures. In *Proceedings of the 2018 IEEE SENSORS*. 1–4.

[14] Christopher W. Fletcher, Ling Ren, Albert Kwon, Marten Van Dijk, and Srinivas Devadas. 2015. Freecursive ORAM: [Nearly] free recursion and integrity verification for position-based oblivious RAM. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'15)*. 103–116.

[15] Christopher W. Fletchery, Ling Ren, Xiangyao Yu, Marten Van Dijk, Omer Khan, and Srinivas Devadas. 2014. Suppressing the oblivious RAM timing channel while making information leakage and program efficiency tradeoffs. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA'14)*. 213–224.

[16] Distributed Management Task Force. 2023. Secured Messages using SPDM Specification. Retrieved 11 November 2024 from https://www.dmtf.org/sites/default/files/standards/documents/DSP0277_1.1.0.pdf

[17] Wilfred Gomes, Slade Morgan, Boyd Phelps, Tim Wilson, and Erik Hallnor. 2022. Meteor lake and arrow lake intel next-gen 3D client architecture platform with foveros. In *Proceedings of the Hot Chips Symposium (HCS'22)*. 1–40.

[18] Donghyun Gouk, Sangwon Lee, Miryeong Kwon, and Myoungsoo Jung. 2022. Direct access, high-performance memory disaggregation with DirectCXL. In *Proceedings of the USENIX Annual Technical Conference (ATC'22)*. 287–294.

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

[20] Andrew Huang. 2002. Keeping secrets in hardware: The microsoft xboxtm case study. In *Proceedings of the International Workshop on Cryptographic Hardware and Embedded Systems*. 213–227.

[21] Jinghan Huang, Jiaqi Lou, Srikar Vanavasam, Xinhao Kong, Houxiang Ji, Ipoom Jeong, Danyang Zhuo, Eun Kyung Lee, and Nam Sung Kim. 2024. HAL: Hardware-assisted load balancing for energy-efficient SNIC-host cooperative computing. In *Proceedings of the International Symposium on Computer Architecture (ISCA'24)*. 613–627.

[22] Michael Hutter and Jörn-Marc Schmidt. 2014. The temperature side channel and heating fault attacks. In *Proceedings of the Smart Card Research and Advanced Applications*. 219–235.

[23] Hypersecu. 2020. HYP2003 Cryptographic Module. Retrieved 11 November 2024 from https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3602.pdf

[24] IBM. 2023. IBM 4770 PCIe Cryptographic Coprocessor Hardware Security Module. Retrieved 11 November 2024 from https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp4558.pdf

[25] Infineon. 2022. Infineon Trusted Platform Module 2.0 SLB 9670 cryptographic module. Retrieved 11 November 2024 from https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3492.pdf

[26] Intel. 2023. How Does the DRAM Caching Work in Memory Mode Using Intel® Optane™ Persistent Memory? Retrieved 11 November 2024 from https://www.intel.com/content/www/us/en/support/articles/000055901/memory-and-storage/intel-optane-persistent-memory.html

[27] Intel. 2023. Intel Xeon CPU Max Series Configuration and Tuning Guide. Retrieved 11 November 2024 from https://cdrdv2-public.intel.com/769060/354227-intel-xeon-cpu-max-series-configuration-and-tuning-guide.pdf
[28] Intel. 2023. Intel® Xeon® CPU Max Series. Retrieved from https://www.intel.com/content/www/us/en/products/docs/processors/xeon/xeon-max-series-product-brief.html
[29] Intel. 2024. Intel® Trust Domain Extensions (Intel TDX). Retrieved 11 November 2024 from https://www.intel.com/content/www/us/en/developer/tools/trust-domain-extensions/overview.html
[30] Phil Isaacs, Thomas Morris Jr, Michael J Fisher, and Keith Cuthbert. 2013. Tamper proof, tamper evident encryption technology. In *Proceedings of the Pan Pacific Symposium*.
[31] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Proceedings of the Ndss*. 12.
[32] Junhyeok Jang, Hanjin Choi, Hanyeoreum Bae, Seungjun Lee, Miryeong Kwon, and Myoungsoo Jung. 2023. CXL-ANNS: Software-hardware collaborative memory disaggregation and computation for billion-scale approximate nearest neighbor search. In *Proceedings of the USENIX Annual Technical Conference (ATC'23)*. 585–600.
[33] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White Paper* 13 (2016).
[34] The kernel development community. 2024. Linux kernel driver APIs - compute express link memory devices. Retrieved 11 November 2024 from https://docs.kernel.org/driver-api/cxl/memory-devices.html
[35] Sanjeev Khushu and Wilfred Gomes. 2019. Lakefield: Hybrid cores in 3D package. In *Proceedings of the Hot Chips Symposium (HCS'19)*. 1–20.
[36] Jonghyeon Kim, Wonkyo Choe, and Jeongseob Ahn. 2021. Exploring the design space of page management for multi-tiered memory systems. In *Proceedings of the USENIX Annual Technical Conference (ATC'21)*. 715–728.
[37] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. 2014. Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors. *ACM SIGARCH Computer Architecture News* 42, 3 (2014), 361–372.
[38] Vladimir Kiriansky, Ilia Lebedev, Saman Amarasinghe, Srinivas Devadas, and Joel Emer. 2018. DAWG: A defense against cache timing attacks in speculative execution processors. In *Proceedings of the International Symposium on Microarchitecture (MICRO'18)*. 974–987.
[39] Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. 2011. Introduction to differential power analysis. *Journal of Cryptographic Engineering* 1 (2011), 5–27.
[40] Miryeong Kwon, Junhyeok Jang, Hanjin Choi, Sangwon Lee, and Myoungsoo Jung. 2023. Failure tolerant training with persistent memory disaggregation over CXL. *IEEE Micro* 43, 2 (2023), 66–75.
[41] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. 2020. Rambleed: Reading bits in memory without accessing them. In *Proceedings of the Symposium on Security and Privacy (SP'20)*. 695–711.
[42] Sunho Lee, Jungwoo Kim, Seonjin Na, Jongse Park, and Jaehyuk Huh. 2022. TNPU: Supporting trusted execution with tree-less integrity protection for neural processing unit. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'22)*. 229–243.
[43] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. 2023. Memtis: Efficient memory tiering with dynamic page classification and page size determination. In *Proceedings of the Symposium on Operating Systems Principles (SOSP'23)*. 17–34.
[44] Baptiste Lepers and Willy Zwaenepoel. 2023. Johnny cache: The end of *DRAM* cache conflicts (in tiered main memory systems). In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'23)*. 519–534.
[45] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-based memory pooling systems for cloud platforms. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23)*. 574–587.
[46] Shang Li, Zhiyuan Yang, Dhiraj Reddy, Ankur Srivastava, and Bruce Jacob. 2020. DRAMsim3: A cycle-accurate, thermal-capable DRAM simulator. *IEEE Computer Architecture Letters* 19, 2 (2020), 106–109.
[47] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading kernel memory from user space. In *Proceedings of the USENIX Security Symposium (Security'18)*. 973–990.
[48] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B. Lee. 2016. Catalyst: Defeating last-level cache side channel attacks in cloud computing. In *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA'16)*. 406–418.
[49] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *Proceedings of the Symposium on Security and Privacy (SP'15)*. 605–622.
[50] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. 2013. Phantom: Practical oblivious computation in a secure processor. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'13)*. 311–324.

[51] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent page placement for CXL-enabled tiered-memory. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'23)*. 742–755.

[52] Thomas S. Messerges. 2000. Securing the AES finalists against power analysis attacks. In *Proceedings of the International Workshop on Fast Software Encryption*. 150–164.

[53] Thomas S. Messerges and Ezzy A. Dabbish. 1999. Investigations of power analysis attacks on smartcards. In *USENIX Workshop on Smartcard Technology (Smartcard 99)*. 151–161.

[54] Motorola. 2019. ASTRO CDEM Motorola Advanced Crypto Engine. Retrieved 11 November 2024 from https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3512.pdf

[55] Seonjin Na, Jungwoo Kim, Sunho Lee, and Jaehyuk Huh. 2024. Supporting secure multi-GPU computing with dynamic and batched metadata management. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'24)*. 204–217.

[56] Seonjin Na, Sunho Lee, Yeonjae Kim, Jongse Park, and Jaehyuk Huh. 2021. Common counters: Compressed encryption counters for secure GPU memory. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'21)*. 1–13.

[57] National Institute of Standards and Technology (NIST). 2019. FIPS 140-3, Security Requirements for Cryptographic Modules. Retrieved 11 November 2024 from https://csrc.nist.gov/pubs/fips/140-3/final

[58] National Institute of Standards and Technology (NIST). 2024. Implementation Guidance for FIPS 140-3 and the Cryptographic Module Validation Program. Retrieved 11 November 2024 from https://csrc.nist.gov/csrc/media/Projects/cryptographic-module-validation-program/documents/fips%20140-3/FIPS%20140-3%20IG.pdf

[59] National Institute of Standards, Technology (NIST), and Morris J. Dworkin. 2015. SHA-3 Standard: Permutation-based hash and extendable-output functions. *Federal Inf. Process. Stds. (NIST FIPS)*, National Institute of Standards. DOI : https://doi.org/https://doi.org/10.6028/NIST.FIPS.202

[60] Sang-Soo Park, KyungSoo Kim, Jinin So, Jin Jung, Jonggeon Lee, Kyoungwan Woo, Nayeon Kim, Younghyun Lee, Hyungyo Kim, Yongsuk Kwon, Jinhyun Kim, Jieun Lee, YeonGon Cho, Yongmin Tai, Jeonghyeon Cho, Hoyoung Song, Jung Ho Ahn, and Nam Sung Kim. 2024. An LPDDR-based CXL-PNM platform for tco-efficient inference of transformer-based large language models. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'24)*.

[61] Intellect Partners. 2023. Migration from Hybrid Memory Cube (HMC) to High-Bandwidth Memory (HBM). Retrieved 11 November 2024 from https://intellect-partners.com/blog/migration-from-hybrid-memory-cube-hmc-to-high-bandwidth-memory-hbm

[62] Rachit Rajat, Yongqin Wang, and Murali Annavaram. 2022. PageORAM: An efficient DRAM page aware ORAM strategy. In *Proceedings of the International Symposium on Microarchitecture (MICRO'22)*. 91–107.

[63] Rachit Rajat, Yongqin Wang, and Murali Annavaram. 2023. Laoram: A look ahead oram architecture for training large embedding tables. In *Proceedings of the International Symposium on Computer Architecture (ISCA'23)*. 1–15.

[64] Rambus. 2024. DDR5 Server DIMM Chipset. Retrieved 11 November 2024 from https://www.rambus.com/memory-interface-chips/ddr5-dimm-chipset/

[65] Realia. 2018. Cryptosec Dekaton. Retrieved 11 November 2024 from https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3329.pdf

[66] Ling Ren, Xiangyao Yu, Christopher W. Fletcher, Marten Van Dijk, and Srinivas Devadas. 2013. Design space exploration and optimization of path oblivious ram in secure processors. In *Proceedings of the International Symposium on Computer Architecture (ISCA'13)*. 571–582.

[67] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. 2007. Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly. In *Proceedings of the International Symposium on Microarchitecture (MICRO'07)*. 183–196.

[68] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. 2020. AIFM: High-performance, application-integrated far memory. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*. 315–332.

[69] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, Jose A. Joao, and Moinuddin K. Qureshi. 2018. Morphable counters: Enabling compact integrity trees for low-overhead secure memories. In *Proceedings of the International Symposium on Microarchitecture (MICRO'18)*. 416–427.

[70] Daniel Sanchez and Christos Kozyrakis. 2013. ZSim: Fast and accurate microarchitectural simulation of thousand-core systems. *ACM SIGARCH Computer Architecture News* 41, 3 (2013), 475–486.

[71] NXP Semiconductors. 2020. NXP Semiconductors JCOP4 P71 cryptographic module. Retrieved 11 November 2024 from https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3746.pdf

[72] Wisekey Semiconductors. 2018. VaultIC420 and VaultIC460. Retrieved 11 November 2024 from https://csrc.nist.gov/CSRC/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp3533.pdf

[73] Sean W. Smith and Steve Weingart. 1999. Building a high-performance, programmable secure coprocessor. *Computer Networks* 31, 9 (1999), 831–860.

[74] Chihun Song, Michael Jaemin Kim, Tianchen Wang, Houxiang Ji, Jinghan Huang, Ipoom Jeong, Jaehyun Park, Hwayong Nam, Minbok Wi, Jung Ho Ahn, and Nam Sung Kim. 2024. TAROT: A CXL SmartNIC-based defense against multi-bit errors by row-hammer attacks. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'24)*. 981–998.

[75] Emil Stefanov, Marten van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path ORAM: An extremely simple oblivious RAM protocol. *Journal of the ACM* 65, 4 (2018), 1–26.

[76] Lisa Su and Sam Naffziger. 2023. 1.1 Innovation for the next decade of compute efficiency. In *Proceedings of the International Solid-State Circuits Conference (ISSCC'23)*. 8–12.

[77] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2023. Demystifying CXL memory with genuine CXL-ready systems and devices. In *Proceedings of the International Symposium on Microarchitecture (MICRO'23)*. 105–121.

[78] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. 2018. VAULT: Reducing paging overheads in SGX with efficient integrity verification structures. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'18)*. 665–678.

[79] Mingtian Tan, Junpeng Wan, Zhe Zhou, and Zhou Li. 2021. Invisible probe: Timing attacks with PCIe congestion side-channel. In *Proceedings of the Symposium on Security and Privacy (SP'21)*. 322–338.

[80] Xconn Technologies. 2024. XConn: CXL Switches for AI. Retrieved 11 November 2024 from https://www.youtube.com/watch?v=oCldo3GgJKg

[81] Daniel Townley, Kerem Arıkan, Yu David Liu, Dmitry Ponomarev, and Oğuz Ergin. 2022. Composable cachelets: Protecting enclaves from cache side-channel attacks. In *Proceedings of the USENIX Security Symposium (Security'22)*. 2839–2856.

[82] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: A high resolution, low noise, l3 cache side-channel attack. In *Proceedings of the USENIX Security Symposium (Security'14)*. 719–732.

[83] Wonsup Yoon, Jisu Ok, Jinyoung Oh, Sue Moon, and Youngjin Kwon. 2023. DiLOS: Do not trade compatibility for performance in memory disaggregation. In *Proceedings of the European Conference on Computer Systems (Eurosys'23)*. 266–282.

[84] Xiangyao Yu, Syed Kamran Haider, Ling Ren, Christopher Fletcher, Albert Kwon, Marten Van Dijk, and Srinivas Devadas. 2015. Proram: Dynamic prefetcher for oblivious ram. In *Proceedings of the International Symposium on Computer Architecture (ISCA'15)*. 616–628.

[85] Shougang Yuan, Amro Awad, Ardhi Wiratama Baskara Yudha, Yan Solihin, and Huiyang Zhou. 2022. Adaptive security support for heterogeneous memory on GPUs. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'22)*. 213–228.

[86] Xiaotong Zhuang, Tao Zhang, and Santosh Pande. 2004. HIDE: An infrastructure for efficiently protecting information leakage on the address bus. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'04)*. 72–84.

Received 21 June 2024; revised 11 October 2024; accepted 26 October 2024