Efficient Security Support for CXL Memory through Adaptive Incremental Offloaded (Re-)Encryption

Chuanhan Li
UC-Santa Cruz
Santa Cruz, CA, USA
cli346@ucsc.edu

Jishen Zhao
UCSD
San Diego, CA, USA
jzhao@ucsd.edu

Yuanchao Xu
University of California, Santa Cruz
Santa Cruz, CA, USA
yxu314@ucsc.edu

Abstract

Current DRAM technologies face critical scaling limitations, significantly impacting the expansion of memory bandwidth and capacity required by modern data-intensive applications. Compute eXpress Link (CXL) emerges as a promising technology to address these limitations, enabling efficient cache-coherent memory expansion through direct connections between processors and CXL memory devices. Despite its potential, broad adoption of CXL memory in public cloud computing introduces substantial security challenges. Trusted Execution Environments (TEEs), such as Intel SGX/TDX and AMD SEV, provide robust protection for data integrity and confidentiality in cloud environments, complemented by CXL Integrity and Data Encryption (CXL IDE), which employs XTS encryption and Galois/Counter Mode (GCM) for secure message transmission.

However, this approach incurs significant performance overhead due to the latency of XTS encryption on memory-intensive workloads. To mitigate this, we propose Adaptive Incremental Offloaded (Re-)Encryption (AIORE), an adaptive security framework combining Counter (CTR) and XTS encryption. AIORE dynamically selects encryption schemes based on page access frequency, implements incremental and offloaded re-encryption strategies, and leverages memory node computation to reduce overheads. Evaluation with Gem5 across diverse benchmarks reveals that AIORE significantly reduces security overhead by 62.8% on average and maintains overhead within 3.7% relative to an insecure baseline.

ACM Reference Format:
Chuanhan Li, Jishen Zhao, and Yuanchao Xu. 2025. Efficient Security Support for CXL Memory through Adaptive Incremental Offloaded (Re-)Encryption. In *58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25), October 18–22, 2025, Seoul, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3725843.3756119

1 Introduction

Current DRAM technologies are approaching their scaling limits, presenting substantial challenges in increasing bandwidth and capacity at a low cost. Compute eXpress Link (CXL) [20] is an open standard developed by a joint effort of hardware manufacturers and cloud computing providers [10, 38, 55–57, 65, 66], aiming to provide low-latency, cache-coherent interconnects between processors, memory devices, and accelerators through load/store commands. A processor can connect directly with CXL memory devices using CXL links to efficiently expand memory bandwidth and capacity, offering a promising solution to solve DRAM scaling challenges.

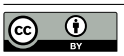
Securing CXL memory and links is crucial for their widespread adoption in public cloud computing, which is inherently susceptible to diverse attacks. Public clouds extensively utilize enclaves from Trusted Execution Environments (TEEs) like Intel SGX/TDX and AMD SEV to ensure the confidentiality and integrity of user data with minimal reliance on cloud providers [6–8, 12, 13, 24, 28, 29, 31, 35, 37, 39, 42, 45, 59, 64, 69, 75, 78–80, 83]. However, CXL memory presents new challenges to existing TEE solutions. First, current TEEs support only a single server node, whereas CXL memory connects multiple nodes. Second, the CXL link introduces a new attack surface that existing TEE frameworks do not cover. Integrating security support for CXL with TEEs already in place for public cloud computing is essential to address these vulnerabilities effectively.

CXL 2.0 introduces CXL Integrity and Data Encryption (CXL IDE) [21], integrating with counterless TEEs (employing XEX Tweakable Block Cipher with Ciphertext Stealing, XTS), TEE IO [5, 41], and the TEE Device Interface Security Protocol (TDISP) [58] to enhance the security of CXL memory. This approach utilizes Galois/Counter Mode (GCM) [51] to encrypt and authenticate messages across the CXL link to guarantee confidentiality, integrity, replay protection, and uniqueness. XTS TEEs on the host CPU securely provide an XTS encryption key to encrypt data before transferring them from the host to the CXL memory and to decrypt the XTS-encrypted ciphertext upon its return from the CXL memory to the host CPU. Further details are available in Section 2.5.

Although this solution meets security requirements, its XTS decryption incurs significant overhead on memory reads, as its encryption process is integral to the critical path of each read. Given that CXL memory is commonly utilized for memory-intensive applications, the XTS TEE and CXL IDE result in substantial overheads for these applications, which can reach up to 14.9% over the insecure case (See Section 6.1). This substantial overhead may hinder the widespread adoption of secure CXL memory.

The state-of-the-art XTS TEE and CXL IDE solution ignores the potential advantages of Counter (CTR) mode encryption. On a counter cache hit, CTR mode enables parallel computation of the One-Time Pad (OTP) and fetching of the required ciphertext cacheline. Upon receiving the ciphertext, a simple XOR operation with the OTP completes the decryption, resulting in latency comparable to an insecure scenario. However, on a counter cache miss, CTR encryption incurs higher bandwidth usage and latency similar to or greater than XTS encryption [82].

We observe two inefficiencies in using CTR TEE and CXL IDE. First, using CTR encryption for all pages reduces counter cache performance, as some cold pages have limited reuse of their cached



MICRO '25, Seoul, Republic of Korea

ACM ISBN 979-8-4007-1573-0/25/10
https://doi.org/10.1145/3725843.3756119

counters, yet these counters pollute the counter cache and reduce overall CTR TEE performance. Second, using a more compact split counter increases counter cache hit rates but requires more frequent per-page re-encryption on counter overflows [87], leading to increased overhead in CXL memory due to the extended link-layer transmission latency from additional data reads and rewrites.

Based on these observations, we introduce Adaptive Incremental Offloaded (Re-)Encryption (AIORE) to maximize the advantages of using CTR TEE for securing CXL memory. AIORE employs three designs: First, per-page adaptive encryption dynamically selects between two encryption methods (XTS or CTR) for each page based on its access frequency to optimize latency, counter cache usage, and bandwidth. Second, incremental re-encryption integrates with program-inherent accesses and encryption/decryption processes to efficiently switch encryption modes or manage per-page re-encryption due to a split counter overflow, significantly reducing re-encryption overhead. Third, offloaded re-encryption transfers incomplete re-encryption tasks to the memory node when a program does not access all cachelines within a page over a certain period, eliminating the overhead on the host and additional data transfer.

We implement and evaluate our solution using the Gem5 simulator [32], comparing it with 7 baselines across 13 workloads from SPEC2017, SPEC2006, and graph applications. The results demonstrate that AIORE reduces security overhead by on average 62.8% compared to the baselines and incurs only 3.2% and 4.3% overhead compared to insecure CXL in single-core and multi-core scenarios.

In summary, the contributions of the works include:

- We conduct a systematic analysis of potential solutions for securing CXL memory from security and performance perspectives.
- We introduce Adaptive Incremental Offloaded (Re-)Encryption (AIORE), which effectively combines CTR and XTS encryption with memory node computation capabilities to improve the efficiency of secure CXL memory.
- Our implementation and evaluation of AIORE demonstrate that it incurs only an average 3.7% overhead relative to insecure CXL, offering an efficient CXL security support.

2 Background

2.1 Compute eXpress Link (CXL)

Building upon PCIe's physical layer, the CXL standard articulates three distinct protocols: CXL.io, CXL.cache, and CXL.mem [10, 57, 65, 66, 70]. CXL.io uses protocol features of standard PCIe, such as transaction-layer packets and data-link-layer packets. PCIe serves as the industry standard for a high-speed serial interface between a CPU and I/O devices. Each lane in PCIe 5.0 supports 32 GT/s. CXL.cache and CXL.mem enable the device to access the CPU's memory and the CPU to access the device's memory, respectively.

The CXL.mem protocol enables the direct connection of memory devices to the host CPU, improving both memory capacity and bandwidth [70, 73]. With support from the host CPU and the CXL controller, this protocol allows for the transparent exposure of CXL memory as remote NUMA node memory, as shown in Figure 1. Data in CXL memory can be directly transferred to the host CPU's

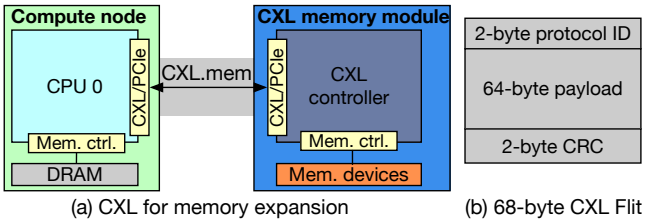


Figure 1: (a) CXL for memory expansion. (b) 68-byte CXL Flit.

Last-Level Cache (LLC) through regular load/store instructions, effectively enhancing memory bandwidth and capacity.

CXL multiplexes its three protocols at the PCIe physical layer. The transfer unit for each protocol is a Flit (Flow-Control Unit). CXL 1.0, 1.1, and 2.0 specifications define a 68-byte Flit [22, 23, 68]. CXL 3.0 specification introduces additional 256-byte Flits [22]. Each 68-byte Flit comprises a 2-byte protocol-ID, a 64-byte payload, and a 2-byte CRC (Cyclic Redundancy Check) protecting the payload (Figure 1 (b)).

2.2 CXL Integrity and Data Encryption

CXL 2.0 introduces Integrity and Data Encryption (IDE)[21], enabling secure end-to-end communication between CXL ports and trusted components. It ensures confidentiality, integrity, uniqueness, and replay protection at the Flit level using 256-bit AES in Galois Counter Mode (AES-GCM)[51]. Figure 2(a) illustrates the detailed computation of ciphertexts and the MAC for a 64-byte cacheline in Skid mode. AES-GCM employs the IV, counter, and encryption key to create a One-Time Pad (OTP), which is then XORed with the plaintext to produce the first 256-bit ciphertext segment. The counter is incremented or computed using a formula securely shared between the sender and receiver, and this synchronized counter is subsequently used to encrypt the second segment. The MAC is generated using a hash subkey derived from the main encryption key, involving operations such as multiplication in the Galois Field and XORing with ciphertexts. Since the OTP takes the synchronized counter as input, it also guarantees message uniqueness, meaning that transferring the same Flit will produce different ciphertexts. Furthermore, as the OTP is address-independent, GCM encryption/decryption latency can be mostly hidden using GCM OTP precomputation[2, 52, 61].

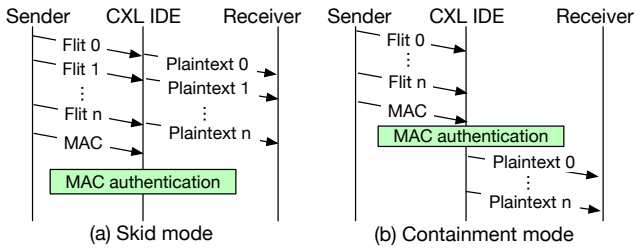


Figure 2: CXL IDE (a) Skid mode and (b) Containment mode.

Unlike previous approaches that generate one GCM MAC per cacheline [9, 62], CXL IDE improves performance by creating a

single GCM MAC for batches of up to 128 cacheline transfers, referred to as a MAC epoch. In the Skid mode of CXL IDE, data is decrypted to plaintext and transmitted to the receiver without awaiting completion of MAC authentication. Conversely, in Containment mode, the plaintext becomes usable only after passing the MAC authentication.

2.3 Memory Encryption

Current TEEs, including AMD SEV and Intel TDX, employ AES-XTS (counterless) memory encryption. Although this encryption does not require additional metadata, except for data address, identical addresses containing the same content will yield the same ciphertext (see Figure 3 (a)).

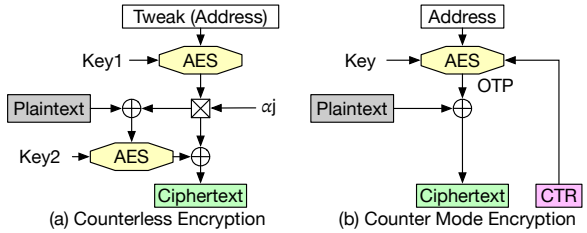


Figure 3: (a) Counterless and (b) Counter Mode Encryption.

In traditional memory encryption schemes, such as Intel SGX1 [39], AES-CTR (counter mode) encryption is employed. Each data block is encrypted and decrypted using an OTP, generated via AES computations that utilize the block's address and an associated counter as inputs (Figure 3 (b)). To ensure ciphertext uniqueness, the counter is incremented with each memory write.

From a performance standpoint, for an LLC miss, CTR encryption hides most latency on counter hits, as the OTP computation can occur concurrently with data retrieval, and the XOR operation incurs trivial latency. However, in applications with irregular access patterns, these benefits are negated due to increased counter misses. Conversely, with XTS encryption, the second AES computation must await the arrival of the ciphertext. Thus, the latency of this second computation is unavoidable.

2.4 Split Counter

Split counter schemes [63, 75, 87] utilize a large major counter alongside a series of minor counters, allowing 64 counters for 64 cachelines in a page to fit into a single 64B cacheline, reducing the space required for counters. Typically, this includes 7-bit and 3-bit minor counters, as illustrated in Figure 4. The counter value for AES encryption is derived by combining the shared major counter with the private minor counter using the formula *Major Counter* ≪ *minor counter bit-width + minor CTR*. Upon a minor counter overflow, the major counter is incremented, requiring the re-encryption of the associated page using the new major counter. This process involves reading, decrypting, re-encrypting, and writing back all cachelines that share this major counter, significantly reducing system performance. Given their limited bit-width, minor counters are more frequent to overflow compared to monolithic counters.

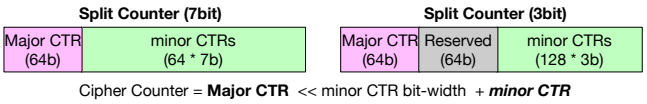


Figure 4: Different split counter schemes

2.5 Baseline and Threat Model

Baseline: Figure 5 illustrates the state-of-the-art baseline utilizing a XTS TEE [4, 40] integrated with CXL IDE [21]. The host CPU employs the XTS TEE to encrypt its local memory using key 1. Similarly, the CXL module encrypts the CXL memory using key 2. The key exchange between the host CPU and CXL module, enabled by TEE IO and TEE TDISP [58], permits the CPU to decrypt ciphertext encrypted with key 2 directly. All message transfers along the CXL link are secured with key 3, which involves encryption and authentication processes executed within the Root Complex of the CXL/PCIe component.

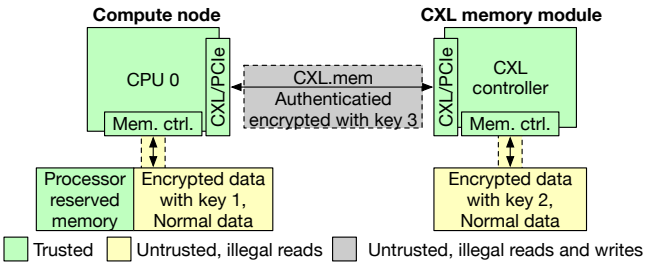


Figure 5: The baseline using XTS TEE and CXL IDE

Threat Model: This baseline adopts the threat model of XTS TEE [4, 40] and CXL IDE [21]. The Trusted Computing Base (TCB) of this system includes the processor, memory controller, CXL controller, CXL/PCIe component, enclave Virtual Machine (VM), and processor-reserved memory, while all other hardware (e.g., off-chip memory) and software (e.g., operating systems and hypervisors) are considered untrusted. Threat scenarios include full adversary control over privileged software, aiming to compromise data confidentiality and integrity. Physical access to the system, which enables data snooping on the CPU–memory bus without the ability to modify the transmitted data.

For the CXL link, an adversary could intercept, tamper with, and replay stale data values. Although important, side-channel attacks [43, 49], CPU bugs [81], and denial-of-service [84] are out of scope of the threat model.

CXL IDE ensures confidentiality, integrity, uniqueness, and freshness, whereas XTS TEE provides solely confidentiality. To protect the CXL link from corruption and replay attacks, a CXL memory read from the host CPU encrypts the 64-byte payload of a 68-byte Flit at the CXL memory module using CXL link key 3 before transmission to the host CPU. Upon receiving, the host CPU decrypts it with key 3. However, the cacheline remains encrypted with key 2 until the host CPU decrypts it again and stores the plaintext in the CPU cache in CXL skid mode. In parallel, the MAC is generated after encryption. Following the transfer of up to 128 messages, the

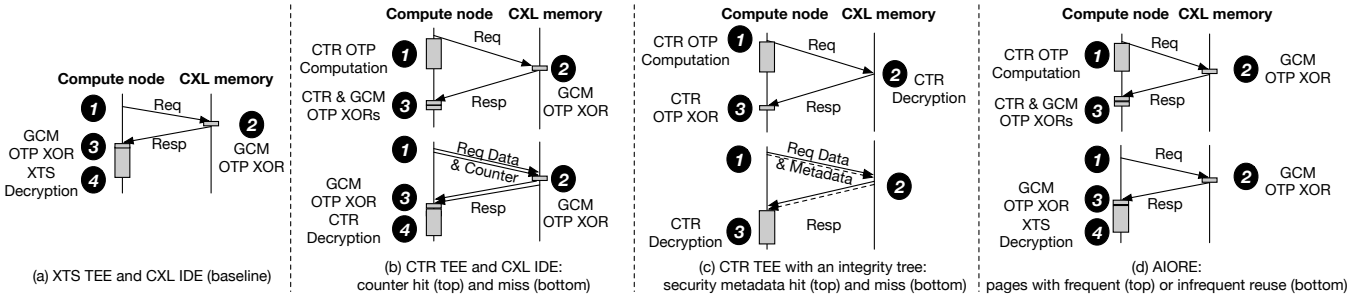


Figure 6: Critical path of an LLC miss for secure CXL memory in different designs under skid mode.

Table 1: Performance and security comparison with different designs for secure CXL memory

Designs	Performance					Security	
	LLC miss latency	Bandwidth consumption	Security metadata updates per write	Counter overflow overhead	Memory space	Confidentiality, Integrity, Replay protection	Uniqueness
XTS TEE and CXL IDE	High	Zero	0	Zero	Zero	Yes	Yes
CTR TEE and CXL IDE	Medium	Medium	1	Low	Small	Yes	Yes
Split CTR TEE and CXL IDE	Low	Low ~ Medium	1	High	Small	Yes	Yes
CTR TEE with an integrity tree	Low	High	Multiple	High	Large	Yes	No
AIORE (our work)	Low	Low	0 or 1	Nearly Zero	Small	Yes	Yes

MAC is sent to the compute node for authentication. A CXL memory write has a similar process to a read, with the difference in the encryption process: data is encrypted with key 2 at the compute node and further encrypted with key 3. The CXL memory module decrypts only the key 3 encryption and stores the ciphertext still encrypted with key 2.

3 Motivation

This section presents our analysis of potential improvements and opportunities for a new solution.

3.1 Analysis of Potential Solutions

A rich set of research has been conducted on improving the performance of TEEs [7, 12, 28, 29, 31, 35, 42, 45, 64, 69, 75, 80, 83]. Our study investigates potential solutions aimed at reducing LLC miss latency in the XTS TEEs and CXL IDE, focusing on approaches such as the CounTeR (CTR) TEE and CXL IDE, the split CTR TEE and CXL IDE, and the CTR TEE with integrity trees.

Figure 6 (a) outlines the critical path operations for an LLC miss in XTS TEE and CXL IDE. Upon an LLC miss, the compute node sends a data fetch request to the CXL memory❶. The CXL memory responds by performing GCM encryption on the XTS-encrypted ciphertext cacheline using a precomputed GCM OTP and generating a MAC from the GCM-encrypted ciphertext ❷. When the Flit carrying the encrypted cacheline reaches the compute node, GCM decryption occurs using the precomputed OTP, followed by XTS decryption to restore the plaintext ❸❹. After a MAC epoch, the MAC is transferred to the compute node for the authentication of all Flits within this epoch.

Although data is XTS-encrypted in CXL memory, transferring the XTS-encrypted ciphertext directly violates the threat model of CXL IDE, which employs the GCM algorithm. This direct transfer fails to ensure uniqueness, exposing it to non-plaintext attacks. In addition, GCM encryption and decryption can be performed effectively by XORing the XTS-encrypted ciphertext with the precomputed OTP, incurring negligible overhead.

CTR TEE and CXL IDE: One possible solution is to replace the counterless TEE with CTR mode TEE. This solution can hide the CTR decryption latency on counter hits, as some counters are cached in the compute node. Figure 6 (b) shows critical path operations of an LLC miss for CTR TEE and CXL IDE. If the counter of the requested cacheline is hit in the compute node, the data fetch and calculation of CTR OTP can be performed in parallel ❶. Afterward, the compute node first performs GCM decryption with precomputed GCM OTP, and then plaintext can be obtained by XORing CTR-encrypted ciphertext with the precomputed CTR OTP ❸. If the counter is missing in the compute node, both the requested data and its counter need to be fetched to perform CTR decryption ❹, exposing similar latency as the baseline.

Split CTR TEE and CXL IDE: This design leverages split counter designs [75, 87] to improve hit rates and reduce bandwidth consumption in CTR TEE and CXL IDE. However, it leads to more frequent counter overflows, requiring the re-encryption of entire pages and increasing high counter overflow overhead.

CTR TEE with an integrity tree: Another alternative solution is to utilize CTR TEE with an integrity tree as the replacement for the entire baseline. CTR mode TEEs equipped with integrity trees ensure the integrity and replay protection by utilizing per-cacheline MACs and an integrity tree derived from encryption counters. This method includes all baseline defenses, except for message uniqueness. In a CXL environment, transferring ciphertext and associated security metadata to the compute node enables integrity verification and replay attack detection without the need for GCM encryption and decryption. Provided the integrity tree's root is securely transmitted, any modifications to ciphertext or security metadata become detectable.

Figure 6 (c) illustrates the critical path operations of this solution. Beyond concealing TEE decryption latency in CTR mode TEEs, this

approach eliminates the need for GCM encryption and decryption. Verification of the MAC and integrity tree can occur either concurrently with or subsequent to plaintext calculation ❸, aligning with CXL IDE defenses. If any security metadata, such as the counter, MAC, or integrity tree node, is missing at the compute node, it must be retrieved from CXL memory.

3.2 Limitations of State-of-the-art Solutions

As analyzed in Section 3.1, CTR encryption can eliminate most security overhead with a high counter cache hit rate. We quantitatively evaluate state-of-the-art counter cache optimizations and hybrid CTR-XTS encryption. Following prior work, our experiments use a dedicated counter cache and memory-intensive benchmarks on Gem5 v24.0.0.1 [32]. All application memory is allocated on CXL memory. The default cache replacement policy is LRU. Further details of our evaluation methodology are provided in Section 5.

3.2.1 Prior Work on Optimizing Counter Cache. A practical approach to improving counter cache hit rates is to use a more compact counter format. However, this increases overhead due to more frequent minor counter overflows, as all cachelines sharing a major counter (e.g., 64 cachelines in a 7-bit split counter) must be read, decrypted, re-encrypted, and written upon a minor counter overflow. During this process, access to the affected cachelines is blocked until re-encryption completes.

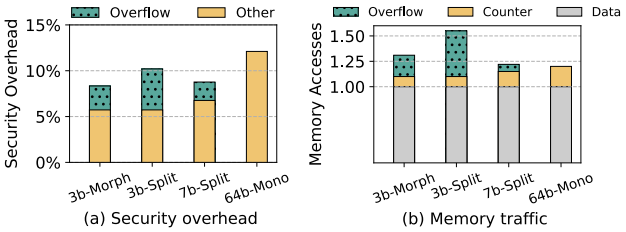


Figure 7: Security overhead (a) and memory traffic (b) of four counter schemes.

We evaluate four counter designs: 3-bit minor counter (3b-Split) [63], 7-bit minor counter (7b-Split) [87], 64-bit monolithic counter (64b-Mono), and 3-bit morphable counter (3b-Morph) [63], which optimizes overflow handling. Figure 7 (a) presents the security overhead (i.e. performance overhead introduced by security functionalities) of these schemes. More compact counters improve counter cache hit rates; however, they significantly increase overflow overhead. For instance, 3b-Split has only 5.8% average other security overhead, but overflows add 4.3%. Figure 7 (b) breaks down memory accesses to data, counters, and overflow-related operations. While 3b-Split reduces counter traffic, it introduces substantial (45% over data access) traffic from handling overflow. The additional traffic reduces available bandwidth and increases contention, further affecting program performance.

Morphable Counter. The morphable counter [63] is designed for integrity trees, which are less effective in securing CXL with State-Of-The-Art (SOTA) TEEs, as state-of-the-art TEEs (e.g., Intel TDX, AMD SEV) have eliminated integrity trees. It still incurs non-trivial a 2.4% overflow overhead and 20.3% overflow traffic compared to the insecure baseline. This is because it relies on two assumptions: (1) the second and third lowest levels of the integrity tree often contain zeros, an assumption that is not applicable under SOTA TEEs; and (2) stream-like access patterns allow grouped counter updates, which are less common in memory intensive workloads on CXL.

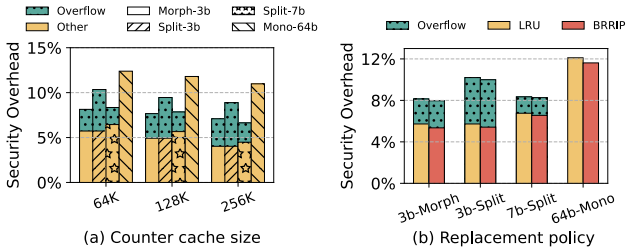


Figure 8: Security overhead in (a) different counter cache sizes and (b) counter cache replacement policy.

Larger Cache and Alternative Cache Replacement Policies. We further evaluate the impact of larger counter cache sizes and the BRRIP replacement policy on counter cache performance. Figure 8 (a) shows the security overhead of various counter formats under different cache sizes, while Figure 8 (b) presents results with the BRRIP policy. Although both approaches reduce counter cache miss rates, overflow overhead and associated traffic remain unchanged, resulting in limited performance gains.

Observation 1. *Counter overflow overhead and associated memory traffic remain key bottlenecks in securing CXL memory.*

3.2.2 Prior Work on Hybrid XTS and CTR Encryption. A recent work, Counter Light [82], proposes adding the 16-byte ECC bus to support hybrid XTS and CTR encryption for improved TEE performance. On an LLC write, the memory controller selects XTS if bandwidth utilization exceeds a threshold and CTR otherwise. The additional 16-byte ECC stores the counter and encryption mode for each cacheline. On an LLC miss, both the 64-byte encrypted data and 16-byte ECC are fetched. If the cacheline uses CTR, the OTP can be cached or computed in parallel with the data fetch on a counter cache hit; otherwise, OTP generation occurs after the ECC arrives.

Counter Light has two main limitations. First, it selects between CTR and XTS encryption on each LLC writeback based on BW utilization to reduce write traffic. However, selecting encryption modes based on access frequency to optimize latency and overall traffic would offer greater performance benefits. Second, it sacrifices compatibility by adding a separate 16-byte ECC bus alongside the 64-byte data bus, which is incompatible with CXL's 68-byte Flit designed for extensibility on future switch-based systems.

To quantify the limitations of selecting encryption modes based on BW utilization during LLC writebacks, we evaluate Counter Light without ECC, incorporating a 64KB counter cache with a 7-bit split counter. Evaluation details are provided in Section 5. In the first experiment, we vary the BW saturation threshold used to switch from default CTR mode to the XTS mode. In the second experiment, we sort pages by access frequency and statically encrypt varying

percentages of the hottest pages with CTR mode, leveraging its efficiency for frequent accesses, while encrypting the rest with XTS.

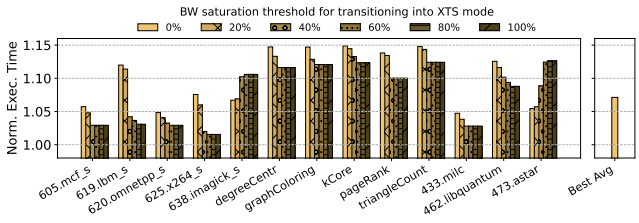


Figure 9: Normalized performance of Counter Light under varying BW saturation thresholds for selecting XTS encryption.

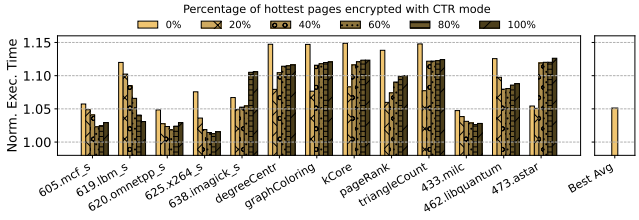


Figure 10: Normalized performance of Counter Light by selecting different percentages of the hottest pages encrypted with CTR mode.

Figure 9 presents the security overhead of Counter Light under different BW thresholds. Performance remains largely unchanged, as encryption mode selection is solely based on BW utilization at each LLC write-back, which does not strategically optimize overall performance. Figure 10 shows the results when different percentages of hottest pages are statically encrypted with CTR. In most benchmarks, the best configuration of this approach yields better performance than the best configuration in the first experiment, showing 28.2% security overhead reduction.

These results indicate that Counter Light's BW-based selection is not strategic, as data is assigned an encryption mode in a non-optimal manner. This highlights the potential of a dynamic, hotness-aware encryption mode selection strategy to better leverage the benefits of hybrid CTR and XTS encryption.

Observation 2. *Selecting XTS and CTR encryption based on access hotness can better exploit the benefits of hybrid encryption.*

4 Adaptive Incremental Offloaded (Re-)Encryption (AIORE)

4.1 Solution Intuition

Based on the two observations in Section 3, our design includes two intuitions as follows.

(1) **Hotness-aware Adaptive Encryption.** Our system dynamically monitors page hotness and initiates re-encryption to adapt encryption modes: hot pages are converted to CTR-encrypted to reduce access latency and improve counter cache hit rates, while cold pages are converted to XTS-encrypted to shrink the counter cache working set and reduce memory traffic from rarely reused counters.

(2) **Incremental and Offloaded Re-encryption.** Page re-encryption, triggered by encryption mode transitions or minor counter overflows, is performed incrementally alongside regular program accesses and encryption/decryption operations, incurring no additional overhead. For pages not fully accessed within a given period, incomplete re-encryption is offloaded to the CXL memory module, avoiding latency on the critical path.

With these two designs, our solution is able to fully exploit performance benefits of hybrid encryption and offloading most of overflow and encryption transition from the critical path without adding extra memory buses.

4.2 Overview

To efficiently materialize our solution, we propose **Adaptive Incremental Offloaded (Re-)Encryption (AIORE)**, which includes three designs: **page hotness tracker**, **incremental re-encryption**, and **offloaded re-encryption**. The **page hotness tracker** dynamically monitors LLC misses and writebacks to determine the appropriate encryption mode for each page and initiates re-encryptions when necessary. For page re-encryption initiated by either an encryption mode transition or a minor counter overflow, re-encryption operations are finished with program inherent accesses and encryption/decryption processes, enabling incremental re-encryption. Specifically, when a program reads a cacheline, it is automatically decrypted using the previous encryption mode and, if applicable, the old counter value. Upon writing back the cacheline, the new encryption mode and, if CTR-encrypted, the new counter value are employed. This approach leverages inherent program memory accesses for seamless re-encryption, avoiding extra operations. However, if a program does not access all cachelines within a page during a certain period, incomplete re-encryption is offloaded to the CXL memory module to complete without adding latency to the critical path.

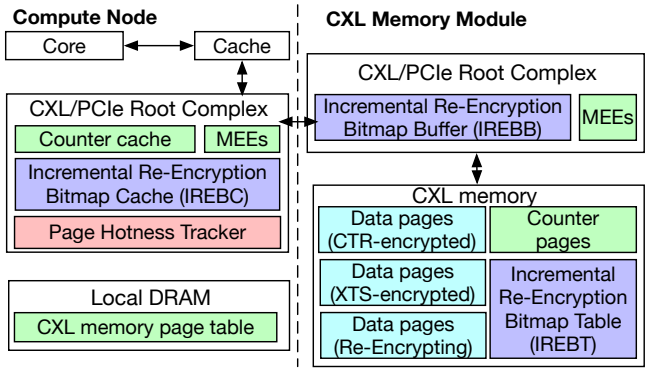


Figure 11: Overview of AIORE Design

Figure 11 depicts the design overview of AIORE. A Page Hotness Tracker is introduced to dynamically assess page accesses based on LLC misses and writebacks, and determine whether a page should

be encrypted using XTS or CTR. To support incremental and offloaded re-encryption, an Incremental Re-Encryption Bitmap Cache (IREBC) is integrated into the compute node's root complex, while an Incremental Re-Encryption Bitmap Buffer (IREBB) is added in the root complex of the CXL memory module, enabling incremental offloaded page re-encryption. Additionally, the Memory Encryption Engines (MEEs) inherently support XTS and GCM encryption in the baseline. We have additionally integrated support for CTR encryption.

CXL memory data pages are classified into three categories: CTR-encrypted, XTS-encrypted, and Re-encrypting. CTR-encrypted and XTS-encrypted pages are defined by the uniform encryption of all cachelines within the page using a single encryption mode. Re-encrypting pages are characterized by a re-encrypting state due to an encryption mode transition or a counter overflow. The CXL memory metadata region contains the counters for CTR-encrypted pages (Counter Pages). In addition, the Incremental Re-Encryption Bitmap Table (IREBT), which stores the encryption mode metadata for all pages, also resides in the metadata space.

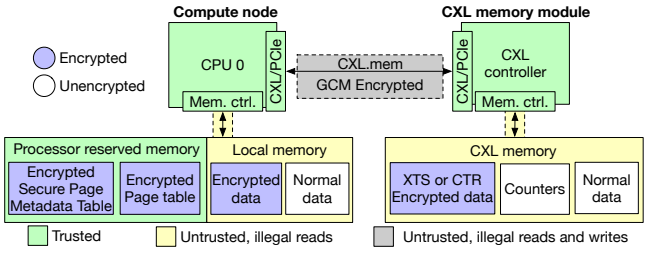


Figure 12: Memory Layout and Metadata

4.2.1 Threat Model and Memory Layout. Our design follows the baseline threat model described in Section 2.5. While it introduces new metadata for counters and their mapping to PFNs, we reuse the dynamic counter mapping proposed by Intel SGX v2 [25], as shown in Figure 12. Specifically, we reuse a secure page metadata table, which is a per-enclave structure similar to a page table and records the mapping from each CTR-encrypted or in-transitioning PFN to its corresponding counters. Each entry stores the physical address of the counter for a PFN and is allocated upon the creation of a CTR-encrypted page and deallocated upon its deletion. Counters are created and deleted together with the allocation and removal of CTR-encrypted pages.

The secure page metadata table resides in trusted processor-reserved memory to prevent unauthorized access, while counters are stored in CXL memory. Our design provides the same confidentiality guarantees of Intel SGX: obtaining ciphertext and counters without knowing the secure encryption key does not compromise data confidentiality. Both the secure page metadata table and the page table are XTS-encrypted with the per-enclave key to ensure confidentiality during transfers between the processor and DRAM.

4.2.2 Workflow. The operations of AIORE are illustrated as follows: Upon an LLC miss, the compute node issues a data read request to the CXL memory module. The CXL memory module retrieves the XTS- or CTR-encrypted ciphertext from its DRAM, packs the ciphertext into a Flit, and then performs AES-GCM encryption and authentication on the Flit. The GCM-encrypted Flit is then transferred to the compute node. The compute node first decrypts the Flit to obtain the XTS- or CTR-encrypted ciphertext, and then invokes XTS or CTR decryption on the ciphertext to recover the plaintext, which is then placed in the cache.

For an LLC write-back request, the compute node first encrypts the dirty cacheline using either XTS or CTR mode, depending on the encryption state. The resulting ciphertext is then packed into a Flit, which undergoes a second layer of encryption using the AES-GCM engine for secure transmission. This GCM-encrypted Flit is then transferred to the CXL memory module. The CXL memory module decrypts the Flit using GCM, and the resulting XTS- or CTR-encrypted ciphertext is stored in device DRAM.

4.3 Incremental Re-Encryption

As AIORE is based on the split counter for improving counter cache performance, re-encrypting a page due to an encryption mode transition or a counter overflow involves decrypting and then re-encrypting all cachelines within it, requiring an additional 64 CXL memory reads and writes. Throughout this transition process, access to the affected page is blocked, preventing any read or write operations until the re-encryption is completed.

To reduce the additional read, write, and blocking overhead associated with a page re-encryption, we introduce an incremental re-encryption design. This approach leverages the encryption and decryption processes inherent in program accesses to finish the re-encryption. Specifically, when a program reads a cacheline, it is automatically decrypted using the previous encryption mode and, if applicable, the old counter value. Upon writing back the cacheline, the new encryption mode and, if CTR-encrypted, the new counter value are employed. Moreover, in cases of silent eviction from the cache, AIORE will encrypt the evicted cacheline using the new mode. Incremental re-encryption significantly reduces the need for additional reads, writes, and the associated blocking during re-encryption.

To support this idea, we design an Incremental Re-Encryption Bitmap Cache (IREBC), which is an LRU cache that records re-encryption states of pages accessed by the host, including both re-encrypting and regular ones, and backs on the in-memory Incremental Re-Encryption Bitmap Table (IREBT) for the entry replacement. For re-encrypting pages, IREBC records the re-encryption states of all cachelines within a page (refer to Figure 13). IREBT also records the regular pages, allowing the determination of the cacheline encryption mode in LLC writeback.

Each IREBC entry has a 36-bit Page Frame Number (PFN), 3-bit re-encryption state. For those re-encrypting pages, an extra 64-bit incremental re-encryption bitmap is added, to specify whether the re-encryption of each cacheline in this page has completed or not.

We adopt a 3-bit minor counter scheme [63] with a slight modification. Instead of using a single major counter shared across 128 3-bit minor counters, each counter cacheline has two major counters, each associated with 64 3-bit minor counters. This design allows two pages linked to the same counter cacheline to re-encrypt independently. Each counter cache entry holds a 36-bit PFN for the first page; the first 32 bytes of the cacheline contain counters for

the first page, with the remaining 32 bytes allocated to the second page.

Additionally, two unused bits in the CXL memory PTE are used to indicate the encryption states, enabling the skipping of IREBC searches for LLC misses on regular pages.

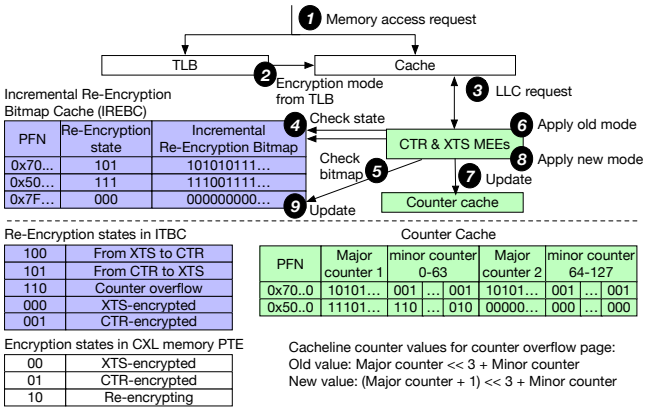


Figure 13: Design and workflow of incremental re-encryption

4.3.1 Handling XTS- or CTR-encrypted pages. ❶ The core sends memory requests to the TLB and cache in parallel. Upon an LLC miss ❸, the page encryption state is retrieved from the TLB result ❷. Upon a cache writeback, the encryption state is obtained through IREBC ❹. The PFN of the page to which the write-back cacheline belongs is used to find the corresponding entry in the IREBC; then, the page re-encryption state is extracted from the entry. If the entry is not found in IREBC (i.e., an IREBC miss), a read is performed to retrieve the missed entry from the Incremental Re-Encryption Bitmap Table (IREBT). If the page is not re-encrypting, access proceeds using either XTS or CTR encryption/decryption, as specified by the encryption mode of the page.

4.3.2 Handling incremental transition between XTS and CTR. If a page is decided to transition between encryption modes, the IREBC entry of the page is modified with the corresponding re-encryption state, and all bitmap values are initialized to 0 to denote an incomplete re-encryption. The PTE's encryption state for this page is updated to "re-encrypting". When the XTS to CTR transition starts, an entry that records the major counter and all minor counters is created with all initial values of 0.

To determine a page's transition from XTS to CTR or XTS to CTR, the system checks the re-encryption state of the IREBC entry ❹. For read accesses, the system also verifies whether the cacheline has completed its re-encryption ❺. If completed, the cacheline is decrypted using the new encryption mode; if not, it continues with the old mode ❻. Write accesses and silent evictions also involve the check of cacheline re-encryption bitmap ❺. The new encryption mode is applied for encryption regardless of the completion of the re-encryption ❽; if incomplete, the system also updates the bitmap to indicate the completion of the re-encryption ❾. Upon completion of all re-encryptions for this page, the entry is reserved for LLC writeback, and the page's PTE is updated to reflect the new encryption status, either XTS- or CTR-encrypted. When transitioning to XTS mode, the corresponding counter cache entry for the previous CTR mode is deleted upon completion of the transition.

4.3.3 Handling incremental re-encryption of counter overflow. Upon detecting a minor counter overflow within a page, all cachelines must be decrypted and re-encrypted with a new counter value. Building on prior works that reset all minor counters to zero and increment the major counter to reduce the frequency of overflows, our approach implements these changes incrementally.

The corresponding entry in the IREBC is modified for the affected page, indicating an incomplete re-encryption with all bitmap values set to zero. The PTE's encryption state is updated to "re-encrypting". In the counter cache, original counter values are used for incremental re-encryptions to new values.

To determine whether a page is undergoing re-encryption due to a counter overflow, the system checks its re-encryption state ❹. If a cacheline has completed its re-encryption, a read access is decrypted using the counter new value, computed as $(major\_counter + 1) \ll 3 + minor\_counter$ ❻. During write-back, the cacheline is encrypted with this counter new value, and the minor counter is incremented by one ❼. If the cacheline has not completed its re-encryption, decryption occurs using the old counter value, $major\_counter \ll 3 + minor\_counter$ ❻. For write-backs or silent evictions, the minor counter is reset to zero ❼, and encryption is conducted with the updated value ❽. Subsequently, the corresponding bitmap bit is marked as completed ❾. Once all bits for the page are marked as completed, the IREBC entry is removed, the major counter is incremented by one, and the page's PTE is updated to reflect its new status as CTR-encrypted.

In AIORE, CXL memory pages are initially encrypted using XTS. While there is a potential to dynamically choose or allow the programmer to specify the initial encryption mode, we do not explore this optimization due to space limitations.

4.4 Offloaded Re-Encryption

Although leveraging incremental re-encryptions that utilize inherent program access and encryption/decryption processes could significantly mitigate overhead and bandwidth consumption, incomplete cacheline access within a certain period may increase IREBC entries, reducing the benefits of the incremental re-encryption.

To address this, we propose offloading the re-encryption to the Root Complex at the CXL memory module when a page's re-encryption remains unfinished for a certain period. Subsequently, any remaining unfinished cachelines are processed within the CXL memory module. This approach effectively hides the latency of the re-encryption process from the critical path and does not demand substantial architectural resources. This is feasible because CXL memory modules are already equipped with memory encryption engines capable of performing encryption and decryption tasks. Moreover, the latency associated with accessing CXL memory from the CXL memory module is considerably lower than that from the compute node, further enhancing efficiency.

To support our design, we introduce the Incremental Re-Encryption Bitmap Buffer (IREBB), which records pages undergoing offloaded re-encryptions (as shown in Figure 14). Each entry in the IREBB is similar to the entry in the IREBC, but additionally includes a 256-bit split counter specific to the page. A difference of the IREBB

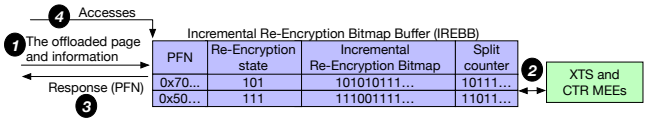


Figure 14: Offloaded Re-Encryption at CXL memory module

is that it operates as a first-come-first-out buffer, which obviates the need for searching entries.

Figure 14 details the offloaded re-encryption workflow. When an entry is evicted from the IREBC in the compute node, it is transferred along with the corresponding counter cacheline to the IREBB ❶. The XTS and CTR MEEs retrieve entries from the IREBB, sequentially processing each not re-encrypted cacheline and updating the IREBB when they are idle ❷. Once all cachelines of a page are processed, a response containing the PFN is sent back to the compute node, which then updates the PTE's encryption state and the associated counter cache entry ❸. If an access is attempted to a page currently being processed (i.e., the first entry in the IREBB), it is temporarily blocked until the update completes ❹. This access blocking mechanism simplifies the concurrency design and marginally impacts performance, as the offloaded pages are those that are infrequently used.

4.5 Page Hotness Tracker

CTR encryption is suitable for frequently accessed pages, while XTS encryption is preferable for pages accessed less frequently. To effectively select encryption modes based on page activities, we propose monitoring page "hotness" to determine the appropriate encryption method and initiate transitions between the two modes as needed. We define a "hot page" as one that receives LLC misses and write-backs within a specified time frame, suggesting higher access rates in the near future. It only monitors LLC misses and write-backs, as decryption only occurs when a cacheline is fetched from CXL memory, and encryption is applied during write-backs from the host to the device. Cache hits and silent evictions are not counted.

When a page is identified as a "hot page," we initiate a transition to convert it to CTR encryption if it is not CTR-encrypted. Conversely, when a page is classified as a "cold page," we transition it to XTS encryption if it is not currently XTS-encrypted. Considering the overhead of transition re-encryption, we employ a higher threshold (i.e. hotness threshold) to categorize a page as hot and a lower threshold (i.e. coldness threshold) to define it as cold, reducing unnecessary re-encryptions.

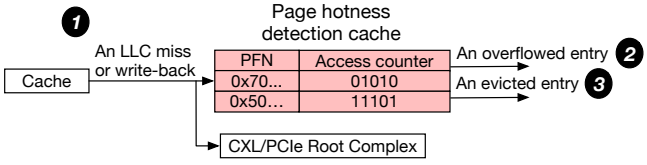


Figure 15: Page hotness tracker at host CPU

To support this idea, we propose a page hotness tracker with Least Recently Used (LRU) replacement policy. Each entry has a 36-bit PFN and a 5-bit access counter that is initialized to 0 when a new entry is created (see Figure 15).

The workflow is shown in Figure 15. When an LLC read miss or write-back occurs, the PFN is extracted, and the access counter for the corresponding entry in the tracker is incremented in parallel with processing the access ❶. If a page's access counter overflows (i.e., reaches the hotness threshold) and the page is currently XTS-encrypted, an interrupt is issued to transition the encryption mode to CTR. This process includes changing the page's encryption state to "re-encrypting" and creating entries in the counter cache and the IREBC with the appropriate re-encryption state ❷. When an entry is evicted from this cache, the system checks the access counter. If the counter value is below the threshold for transitioning to XTS encryption (coldness threshold) and the page is not currently XTS-encrypted, an interrupt is issued to transition the encryption mode to XTS ❸.

To make the page hotness tracker adaptive to varying workloads, the system dynamically adjusts the hotness threshold to maintain a high counter cache hit rate. It periodically monitors the counter cache hit rate: if the hit rate exceeds a predefined threshold, the hotness threshold is slightly reduced to increase the number of CTR-encrypted pages; if the hit rate falls below the threshold, the hotness threshold is slightly increased to reduce the number of CTR-encrypted pages. Although dynamically selecting the hit rate threshold may further optimize performance, in our evaluation a fixed threshold (95%) is used. The hot and cold thresholds are initially set to 16 and 8.

4.6 Area Overhead

AIORE imposes small storage overhead, as detailed in Table 2. On the host side, the root complex is equipped with an IREBC, a page hotness detection cache and a 64KB 8-way counter cache, collectively requiring 67496 bytes of on-chip storage. On the device side, the CXL controller includes an additional 2456 bytes of on-chip storage for IREBB. In total, the combined added storage for both the host and device amounts to 69952 bytes. Every two consecutive CTR-encrypted 4KB pages require a 64-byte counter cacheline, which is only on average 0.78% of the data size.

Table 2: Area overhead for each host node per CXL module.

Host Side	Entry size (bits)	# of entries	Size (bytes)
IREBC	102	128	1632
Page Hotness Detection Cache	41	64	328
Counter Cache	512	1024	65536
Device Side	**Entry size (bits)**	**# of entries**	**Size (bytes)**
IREBB	614	32	2456

A host can connect to multiple CXL modules. For the compute node, the number of entries in the proposed architecture can be extended to efficiently support our design. We estimate a space overhead of 66KB per CXL module. An Intel Xeon 6th processor can connect to up to 8 CXL modules [67], resulting in a maximum of 528KB overhead. This overhead resides in the CXL/PCIe root complex, allowing simple scaling without modifications to core/cache for multiple CXL modules. However, prior research on shared and private caches suggests that it is unnecessary to linearly

increase the number of entries per CXL module to maintain performance [27, 44]. Similarly, each CXL memory module can connect to multiple hosts, adding up to 2.5KB per host. Notably, both host and device side space overhead are independent of the memory capacity, as the size of the IREBB can be adjusted accordingly to match the MEEs throughput. On the host side, higher CXL memory capacity may lead to more LLC misses, which would increase the performance improvement of our work over prior works.

4.7 Power Overhead

We use CACTI [77] to estimate the power overhead of AIORE's additional security-related on-chip storage and encryption engine operations. The added storage contributes 60.41 mW of leakage power, which accounts for an additional 0.83% of the power consumed by the existing on-chip cache.

AIORE has the similar power consumption for supporting encryptions, because existing CXL TEEs are already equipped with AES-XTS and AES-GCM engines for implementing CXL IDE. Our design introduces no additional encryption on the host side due to incremental re-encryption. On the device side, offloaded re-encryption adds some overhead, but it accounts for only a small fraction of the total encryption workload (approximately 4.1% in our experiments).

4.8 Security Analysis

AIORE adopts the baseline's threat model, introduces no new attack surfaces, and does not compromise the security properties of the baseline. All new components are integrated into the trusted computing base of the baseline. AIORE maintains security guarantees that are equivalent to the baseline.

For confidentiality, data in memory is encrypted using either XTS or CTR mode encryption, preventing attackers from stealing sensitive information. When data is transferred through the link, additional GCM encryption further increases the complexity for attackers, enhancing transmission security.

Besides, the integrity of transferred data is ensured by the MAC generated through the GCM authentication algorithm, computing over all the Flits in a MAC epoch. With GHASH and GMAC, if an attacker attempts to replay, reorder, add, or delete Flits, the attempt will fail. Altering any single Flit will result in a MAC mismatch, effectively preventing malicious modifications.

In addition to data security, metadata security is also protected. The host transmits additional control messages, including offloaded re-encryption requests, re-encryption completion notifications, and metadata Flits such as IREBC entries and counter cachelines. These Flits use the same encryption and authentication procedures as data Flits.

AIORE's adaptive encryption introduces varying access latencies, which may lead to potential covert or side channels; however, industry TEEs and CXL IDE exclude timing-based channels from their threat models. Our design follows the same assumptions. There is no evidence that whether our approach leaks more information than prior CTR-based designs, as previous work may also reveal hot addresses and access patterns through counter cache hits.

4.9 Discussion

AIORE can support tiered memory configurations. If local DDR acts as a cache for CXL memory, the memory controller must encrypt data during write-back from cache to local memory, as local memory is vulnerable to potential attacks. As for the flat mode, where local DDR and CXL memory are treated as a single memory space, data migration (such as page promotion) also incurs re-encryption overhead. This is because the encryption mode relies on the memory address as a tweak, and address changes require re-encryption. In this case, AIORE's offloaded re-encryption capability can be used before data transmission, so that the data does not need to be re-encrypted by the host memory controller.

5 Evaluation Methodology

Table 3: Simulator parameters

Processor Configuration	
CPU	4-core, OOO, x86-64, 4.00 GHz
L1 TLB	64 entries, 4-way, Access latency: 1 cycle
L2 TLB	1536 entries, 12-way, Access latency: 9 cycles
L1 Cache	64 KB, 8-way, 64 B block, Access latency: 2 cycles
L2 Cache	512 KB, 16-way, 64 B block, Access latency: 20 cycles
L3 Cache	8MB, 32-way, 64 B block, Access latency: 30 cycles
Memory Controller and CXL Root Complex Configuration	
Memory Channel	One channel per 4-core
Counter Cache	64 KB, 8-way, 64 B block
IREBC	128 entries, Access latency: 30 cycles
IREBB	32 entries, Access latency: 30 cycles
Page Hotness Detection Cache	8-way LRU cache, 64 entries
AES Operation	56 cycles
XOR Operation	1 cycles
GCM Authentication	40 cycles
CXL Device Memory	DDR5 4400 MHz tRCD/tCL/tRP/tRAS/tWR = 14/14/14/32/30 ns
CXL Round Trip Latency	70 ns [46]

Simulator: We evaluate our design using Gem5 v24.0.0.1 [32]. Table 3 presents the simulation arguments. A dedicated cache is maintained for the counter cache. All AES operations, including XTS en/decryption, CTR and GCM OTP calculation are modeled with a latency of 56 cycles, as reported in prior work [82]. The XOR operation with the OTP in GCM and CTR modes takes 1 cycle, while GCM authentication requires 40 cycles. We modify the host memory controller to connect to the device CXL controller, which is implemented as an independent component connected to DDR4 memory. The CXL transmission round-trip latency is set to 70 ns, as shown in prior research [46].

Workloads: We evaluate our design using workloads with various memory access patterns, following the approach of prior works [28, 82, 86]. These workloads are drawn from the benchmark suites SPEC2017 [19], IBM GraphBIG [54], and SPEC2006 [18]. For each workload, we manually mark the region of interest (ROI) to begin just before the main program logic, excluding input reading and preprocessing. A checkpoint is created at the ROI entry, from which we fast-forward Gem5 and simulate 5 billion instructions to collect results.

Schemes: We compare AIORE with seven other schemes:

(1) XTS TEE and CXL IDE (XTS IDE): Enabling transmission security with CXL IDE and memory security with AES-XTS mode.

(2) Split CTR TEE and CXL IDE (CTR IDE): Another CXL IDE-based TEE solution but replace XTS encryption with CTR encryption, using split counter with 7b minor counter, as it provides the best performance among other counter schemes (see Figure 7).
(3) Counter Light and CXL IDE (CL IDE): This scheme uses the Counter Light scheme [82], which supports switching the cacheline encryption mode between CTR and XTS based on a fixed memory bandwidth usage threshold. Notably, Counter Light requires an ECC bus to transfer an additional 16B of MAC, Parity and counter along with the 64B of data in a single memory access, thereby avoiding an additional memory access for the counter. An OTP memorization table is used to store OTPs and eliminate computation on hits.
(4) Counter Light and CXL IDE without ECC Bus (CL IDE w/o ECC Bus): This scheme removes ECC bus support from CL IDE. We replace its memorization table with a 64K counter cache in the read path, avoiding the mandatory second read of the counter due to the absence of the ECC bus. Both CL schemes are equipped with split counter with 7b minor counter.
(5) Morphable Counter and CXL IDE (MorphCTR): This scheme uses 3-bit minor counters to enhance counter cache efficiency and employs Zero Counter Compression along with Minor Counter Rebasing techniques to mitigate the overflow frequency caused by the reduced counter size.
(6) Morphable Counter and CXL IDE with BRRIP Replacement Policy (MorphCTR w/ BRRIP): This scheme replaces the LRU policy with the BRRIP replacement policy in MorphCTR.
(7) ShieldCXL: An oblivious security memory scheme based on the CXL IDE and XTS mode. Obliviousness is supported by encrypting both payload and Flit metadata, and inserting additional dummy Flits when the link is idle to hide the traffic differences between read and write requests.

6 Evaluation

6.1 End-to-end Results

Figure 16 illustrates the normalized execution time of all 13 benchmarks, along with the average results in single-core and eight-core settings. In the single-core case, the CTR IDE and XTS TEE solutions show, on average, 8.3% and 10.5% higher execution times over Insecure CXL, respectively. CTR outperforms XTS in most benchmarks due to the benefit of counter cache hits. The CL IDE with ECC bus reduces 35.7% security overhead over CTR IDE, as the counter arrives simultaneously with the data thanks to the additional ECC bus. However, its performance is constrained by the hit rate of the memorization table, which fails to hide OTP latency when a table miss occurs. Additionally, the memorization table may raise security concerns as it computes the OTP with separated counter value and data address [82]. CL IDE without ECC bus shows an 8.3% overhead, similar to CTR IDE, as few workloads reach the mode transition threshold. MorphCTR and its BRRIP variant exhibit 8.1% and 7.9% overhead, respectively. Although they demonstrate better counter cache efficiency, their performances are hindered by overflow overhead. ShieldCXL incurs a 15.2% overhead, as it must insert dummy Flits to obscure the access patterns of read and write operations. AIORE outperforms CTR IDE by reducing its 58.6% security overhead in the single-core scenario, as it avoids counter cache pollution and eliminates the overhead of re-encryption.

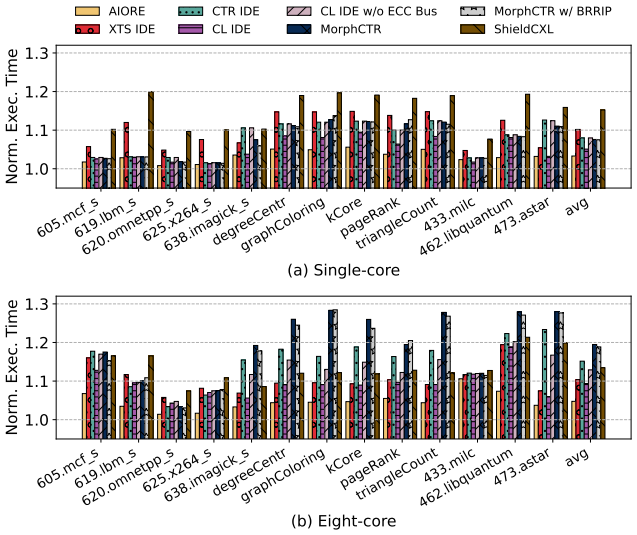


Figure 16: (a) Single-core and (b) Eight-core end-to-end normalized execution times (normalized to the Insecure CXL)

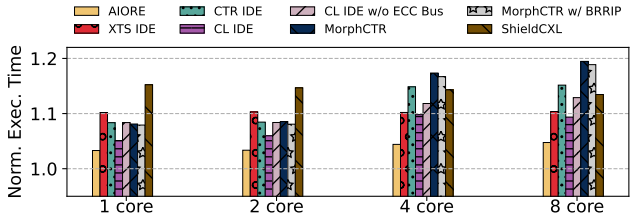


Figure 17: Average end-to-end normalized execution times in 2/4/8 cores (normalized to the Insecure CXL).

In the eight-core scenario, where shared memory bandwidth contention becomes intensive. XTS IDE delivers better performance for most benchmarks than the single-core scenario, inuring 9.9% overhead over Insecure CXL, as it completely eliminates the extra bandwidth consumption associated with retrieving counters. CL IDE benefits from switching cacheline encryption modes from CTR to XTS, incurring 9.3% overhead Insecure CXL. Since this switch occurs at a fixed bandwidth threshold (60%), CL IDE shows similar results to XTS IDE. CL IDE without ECC bus also benefits from XTS mode switching, resulting in a 12.9% overhead. In contrast, the other three CTR based schemes (CTR IDE, MorphCTR, and MorphCTR with BRRIP) experience up to a 28% slowdown, as counter and overflow accesses further compete with data accesses for limited bandwidth. reduces the security overhead by 43.1% and 63.2% compared to CL IDE and CL IDE without ECC bus, respectively, by dynamically selecting the encryption mode and fully leveraging the performance benefits of CTR encryption. Besides, the overhead of

ShieldCXL narrows to 13.5%, as less additional dummy Flits needed due to the busy link.

Figure 17 presents the average performance results for 1-, 2-, 4-, and 8-core scenarios. The data exhibits a trend: as the number of cores increases and bandwidth limitations intensify, XTS IDE gradually outperforms CTR IDE, which requires additional metadata access. The other two CTR based schemes, MorphCTR and MorphCTR with BRRIP show more significant 10.7% and 9.5% performance degradation due to increased memory accesses for handling overflows. In contrast, both Counter Light schemes and AIORE benefit from encryption mode switching. AIORE demonstrates greater improvement in the 4-core and 8-core cases, as Counter Light's switching is solely based on bandwidth consumption, ignoring the potential performance gains from accelerating frequently accessed cachelines. In contrast, AIORE achieves substantial performance gains through the synergy of per-page adaptive encryption and its incremental and offloaded re-encryption mechanisms. In comparison with all other schemes, AIORE incurs a 3.2–4.3% (3.7% on average) security overhead from single- to 8-core settings, reducing the security overhead by 35.3% to 78.3% (62.8% on average).

6.2 Detailed Analysis

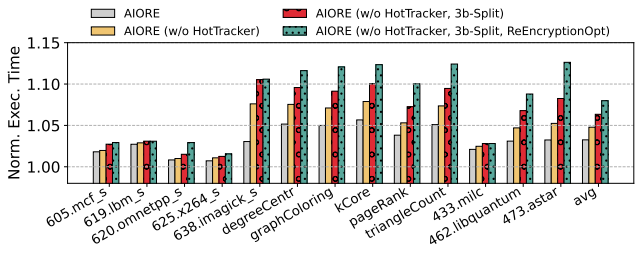


Figure 18: Ablation study of single core (Normalized to the insecure CXL)

Ablation Study: To systematically analyze the source of AIORE's performance gains, we gradually disable different functionalities of AIORE to compare their effects, namely, the page hotness tracker (HotTracker), the split counter with 3-bit minor counters (3b-Split), and incremental and offloaded re-encryption (ReEncryptionOpt). The results are shown in Figure 18. Additionally, to better understand AIORE's performance advantage over CTR mode, we configure AIORE to initially use CTR mode for all pages instead of XTS mode, and make sure the stable system state match that of the XTS case. Page hotness tracking and adaptive encryption approximately reduces 27.7% security overhead over CTR IDE, demonstrating their advantages over CTR mode by eliminating the need to maintain counters for cold pages. Using the 3b split counter rather than the 7b scheme contributes the 15.6% security overhead reduction, due to a reduced counter cache hit rate caused by the less compact counter format. Finally, incremental and offloaded re-encryption contributes the 24.1% security overhead reduction, primarily by removing overflow handling overhead from the critical path.

Page Partition: Figure 19 illustrates the page partitioning results from the page hotness detection, revealing application-specific memory characteristics. For workloads where over 50% of pages are

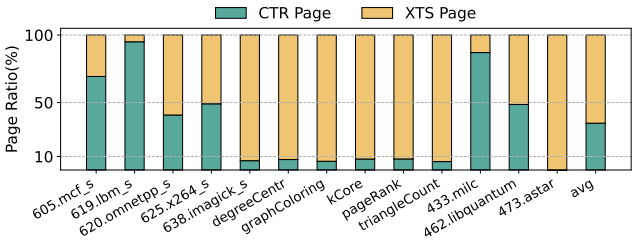


Figure 19: Page partition

classified as CTR pages, such as 605.mcf_s, 619.lbm_s, and 433.milc, AIORE avoids additional metadata overhead by excluding infrequently accessed pages from counter maintenance. This approach increases the likelihood of counter hits for CTR pages and helps prevent counter cache pollution. For benchmarks characterized by low memory reuse and consequently lower CTR page ratios, such as imagick_s, kCore, and astar, AIORE applies XTS to most pages and avoid metadata access. Besides, AIORE effectively captures frequently accessed pages in their memory footprint and accelerates their memory access using CTR mode, thereby achieving improved performance.

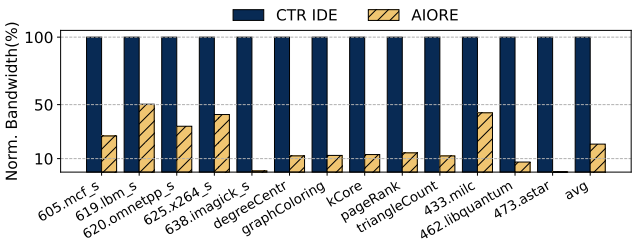


Figure 20: Metadata bandwidth overhead

Metadata Bandwidth: When comparing the bandwidth consumption of metadata reads and writebacks between CTR IDE and AIORE, the reduction in metadata bandwidth usage varies across different workloads, as illustrated in Figure 20. On average, AIORE's metadata bandwidth utilization is approximately 20.8% of that in CTR IDE. This demonstrates AIORE's ability to capture application runtime access patterns and reduce bandwidth consumption through adaptive encryption, incremental and offloaded re-encryption. The sources of bandwidth savings come from a large portion of pages being classified as XTS pages, as seen in workloads like degreeCentr and 638.imagick_s, and also root from reduced re-encryption overhead, such as 620.omnetpp_s and 462.libquantum.

Table 4: Sensitivity Study

IREBC size	Normalized performance	Page Hotness Detection Cache size	Normalized performance
64	1.035	32	1.037
128	1.032	64	1.032
256	1.031	128	1.030

Sensitivity Study We conduct a sensitivity study to evaluate how IREBC and page hotness detection affect AIORE's performance. As shown in Table 4, increasing the IREBC size from 64 to 256 entries slightly reduces overhead from 3.5% to 3.1%. Since re-encryption is off the critical path, its performance impact remains minimal. In addition, the result shows that increasing the hotness detection cache size from 32 to 128 reduces overhead from 3.7% to 3.0%, as it improves hot page identification by tracking access patterns more accurately, enabling better mode-switching decisions.

7 Related Work

Trusted Execution Environments (TEEs): To implement secure hardware enclaves for TEEs, many prior works have proposed enhancing processors by integrating memory encryption and integrity engines [1, 11, 14–16, 30, 34, 47, 48, 60, 63, 72, 76, 85]. While these secure engines provide basic encryption and verification functionality, many designs further support stricter integrity guarantees, such as freshness, using integrity trees like the Merkle Tree [74] and its variants [31, 33, 60, 75]. Many research focus on mitigating the negative impact of overhead from secure engines, like improving the counter locality [35, 45, 53, 80], hiding OTP overhead by pre-computation [2, 3, 50, 52, 61, 69, 83], reducing the counter storage overhead [36, 42, 64], utilizing lightweight encryption algorithm [7, 12]. These works provide parallel solutions to our design and can be integrated into adaptive encryption to speed up the CTR mode encryption. Other works pay attention to reducing the overhead of the integrity tree, including shrinking the integrity levels [42, 75], allowing sharing or transmission of the integrity tree to reduce re-encryption overhead [28, 86]. This work relies on the integrity tree, which has been abandoned in many recent VM-based TEEs and cannot provide the uniqueness guarantee of transmission as discussed before. Midsummer Night's Tree [76] explores metadata locality of hot memory regions to enhance the persistency performance of integrity trees in secure persistent memory.

CXL Security: In the research of CXL TEE, Toleo [26] is the first work to extend TEE into the CXL memory system. Toleo scales memory freshness to tera-scale systems by replacing Merkle trees with trusted smart memory as an extended TCB. It leverages CXL for secure communication and ensures safe, fresh version numbers without recursive integrity checks. ShieldCXL [17] is another work supporting TEE in the CXL environment. It achieves obliviousness for CXL secure memory by inserting dummy Flits and encrypting Flit metadata alongside payload data. However, neither of these works addresses the inherent limitations of CXL IDE-based TEEs. In a related analysis, Stark et al. [71] evaluated the current state of CXL memory safety, highlighting its inflexibility and reliance on coarse-grained mechanisms. The authors propose adopting capability-based systems, for fine-grained and dynamic memory isolation to improve CXL's security and scalability.

8 Conclusion

In this work, we comprehensively analyze security solutions for CXL memory, identifying critical performance bottlenecks associated with current encryption approaches. To effectively secure CXL memory without sacrificing performance, we present Adaptive Incremental Offloaded (Re-)Encryption (AIORE), a novel adaptive encryption framework. By intelligently selecting between CTR and XTS encryption based on access patterns, employing incremental re-encryption strategies, and offloading re-encryption tasks to memory nodes, AIORE substantially reduces encryption overhead. Extensive evaluations demonstrate that AIORE delivers an average performance overhead as low as 3.7%, ensuring efficient and secure adoption of CXL memory technologies in public cloud environments.

Acknowledgments

We sincerely thank the anonymous reviewers from MICRO 2025 and ISCA 2025 for their insightful feedback, which greatly contributed to improving the final version of this paper. Jishen Zhao is supported by SRC JUMP 2.0 Center for Processing with Intelligent Storage and Memory (PRISM).

References

[1] Rahaf Abdullah, Hyokeun Lee, Huiyang Zhou, and Amro Awad. 2024. Salus: Efficient Security Support for CXL-Expanded GPU Memory. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 1–15.

[2] Shaizeen Aga and Satish Narayanasamy. 2017. Invisimem: Smart memory defenses for memory bus side channel. *ACM SIGARCH Computer Architecture News* 45, 2 (2017), 94–106.

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

[4] AMD. 2022. AMD SEV-SNP. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf. Online; accessed Jun, 2022.

[5] AMD. 2024. AMD SEV-TIO: Trusted I/O for Secure Encrypted Virtualization. https://www.amd.com/content/dam/amd/en/documents/developer/sev-tio-whitepaper.pdf. Online; accessed Jun, 2024.

[6] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'keeffe, Mark L Stillwell, et al. 2016. SCONE: Secure linux containers with intel SGX. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 689–703.

[7] Roberto Avanzi. 2017. The QARMA block cipher family. Almost MDS matrices over rings with zero divisors, nearly symmetric even-mansour constructions with non-involutory central rounds, and search heuristics for low-latency s-boxes. *IACR Transactions on Symmetric Cryptology* (2017), 4–44.

[8] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)* 33, 3 (2015), 1–26.

[9] Mihir Bellare and Björn Tackmann. 2016. The multi-user security of authenticated encryption: AES-GCM in TLS 1.3. In *Advances in Cryptology–CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I 36*. Springer, 247–276.

[10] blocksandfiles.com. 2024. Intel sees CXL as rack-level disaggregator with Optane connectivity. https://blocksandfiles.com/2021/08/18/intel-sees-cxl-as-rack-level-disaggregator/. Online; accessed Jun, 2024.

[11] Rick Boivie and Peter Williams. 2012. SecureBlue++: CPU support for secure execution. *IBM, IBM Research Division, RC25287 (WAT1205-070)* (2012), 1–9.

[12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, et al. 2012. PRINCE–a low-latency block cipher for pervasive computing applications. In *Advances in Cryptology–ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings 18*. Springer, 208–225.

[13] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. 2016. Securekeeper: Confidential zookeeper using intel sgx. In *Proceedings of the 17th International Middleware Conference*. 1–13.

[14] David Champagne and Ruby B Lee. 2010. Scalable architectural support for trusted software. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 1–12.

[15] Xiaoxin Chen, Tal Garfinkel, E Christopher Lewis, Pratap Subrahmanyam, Carl A Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan RK Ports. 2008. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. *ACM SIGOPS Operating Systems Review* 42, 2 (2008), 2–13.

[16] Siddhartha Chhabra, Brian Rogers, Yan Solihin, and Milos Prvulovic. 2011. SecureME: a hardware-software approach to full system security. In *Proceedings of the international conference on Supercomputing*. 108–119.

[17] Kwanghoon Choi, Igjae Kim, Sunho Lee, and Jaehyuk Huh. 2024. ShieldCXL: A Practical Obliviousness Support with Sealed CXL Memory. *ACM Transactions on Architecture and Code Optimization* (2024).

[18] Standard Performance Evaluation Corporation. 2006. SPEC CPU 2006. https://www.spec.org/cpu2006/. Online; accessed August, 2020.

[19] Standard Performance Evaluation Corporation. 2017. SPEC CPU 2017. https://www.spec.org/cpu2017/. Online; accessed August, 2020.

[20] CXL. 2024. Compute Express Link. https://computeexpresslink.org/. Online; accessed Jun, 2024.

[21] CXL. 2024. Compute Express Link® (CXL®): Link-level Integrity and Data Encryption (CXL IDE). https://computeexpresslink.org/webinars/compute-express-link-cxl-link-level-integrity-and-data-encryption-cxl-ide-333/.

[22] CXL. 2024. CXL 3.1 Specification. https://computeexpresslink.org/wp-content/uploads/2024/02/CXL-3.1-Specification.pdf. Online; accessed Jun, 2024.

[23] Debendra Das Sharma, Robert Blankenship, and Daniel Berger. 2024. An Introduction to the Compute Express Link (CXL) Interconnect. *Comput. Surveys* 56, 11 (2024), 1–37.

[24] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2020. Oblivious coopetitive analytics using hardware enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–17.

[25] Vijay Dhanraj, Harpreet Singh Chawla, Tao Zhang, Daniel Manila, Eric Thomas Schneider, Erica Fu, Mona Vij, Chia-Che Tsai, and Donald E Porter. 2025. Adaptive and Efficient Dynamic Memory Management for Hardware Enclaves. *arXiv preprint arXiv:2504.16251* (2025).

[26] Juechu Dong, Jonah Rosenblum, and Satish Narayanasamy. 2024. Toleo: Scaling Freshness to Tera-scale Memory using CXL and PIM. *arXiv preprint arXiv:2410.12749* (2024).

[27] Haakon Dybdahl and Per Stenstrom. 2007. An adaptive shared/private nuca cache partitioning scheme for chip multiprocessors. In *2007 IEEE 13th International Symposium on High Performance Computer Architecture*. IEEE, 2–12.

[28] Erhu Feng, Dong Du, Yubin Xia, and Haibo Chen. 2023. Efficient Distributed Secure Memory with Migratable Merkle Tree. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 347–360.

[29] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. 2021. Scalable Memory Protection in the {PENGLAI} Enclave. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*. 275–294.

[30] Christopher W Fletcher, Marten van Dijk, and Srinivas Devadas. 2012. A secure processor architecture for encrypted computation on untrusted programs. In *Proceedings of the seventh ACM workshop on Scalable trusted computing*. 3–8.

[31] Alexander Freij, Huiyang Zhou, and Yan Solihin. 2021. Bonsai Merkle Forests: Efficiently Achieving Crash Consistency in Secure Persistent Memory. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*. 1227–1240.

[32] Gem5. 2024. Gem5 24.0.0.1. https://github.com/gem5/gem5. Online; accessed Jun, 2024.

[33] Shay Gueron. 2016. A memory encryption engine suitable for general purpose processors. *Cryptology ePrint Archive* (2016).

[34] Benjamin Holmes, Jason Waterman, and Dan Williams. 2024. SEVeriFast: Minimizing the root of trust for fast startup of SEV microVMs. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 1045–1060.

[35] Weizhe Hua, Muhammad Umar, Zhiru Zhang, and G Edward Suh. 2022. MGX: Near-zero overhead memory protection for data-intensive accelerators. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 726–741.

[36] Ruirui Huang and G Edward Suh. 2010. Ivec: off-chip memory integrity protection for both security and reliability. *ACM SIGARCH Computer Architecture News* 38, 3 (2010), 395–406.

[37] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2018. Ryoan: A distributed sandbox for untrusted computation on secret data. *ACM Transactions on Computer Systems (TOCS)* 35, 4 (2018), 1–32.

[38] SK Hynix. 2024. SK hynix Develops DDR5 DRAM CXLTM Memory to Expand the CXL Memory Ecosystem. https://news.skhynix.com/sk-hynix-develops-ddr5-dram-cxltm-memory-to-expand-the-cxl-memory-ecosystem/. Online; accessed Jun, 2024.

[39] Intel. 2022. Intel Software Guard Extensions. https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html. Online; accessed Jun, 2022.

[40] Intel. 2023. Intel Trust Domain Extensions (Intel TDX). *https://www.intel.com/content/www/us/en/developer/articles/technical/intel-trust-domain-extensions.html* (2023).

[41] Intel. 2024. Intel® TDX Connect Architecture Specification. https://cdrdv2-public.intel.com/773614/intel-tdx-connect-architecture-specification.pdf. Online; accessed Oct, 2024.

[42] Jungrae Kim, Michael Sullivan, Seong-Lyong Gong, and Mattan Erez. 2015. Frugal ecc: Efficient and versatile memory error protection through fine-grained compression. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[43] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre attacks: Exploiting speculative execution. *arXiv preprint arXiv:1801.01203* (2018).

[44] Hyunjin Lee, Sangyeun Cho, and Bruce R Childers. 2011. CloudCache: Expanding and shrinking private caches. In *2011 IEEE 17th International Symposium on High Performance Computer Architecture*. IEEE, 219–230.

[45] 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 *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 229–243.

[46] Huaicheng Li, Daniel S Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, et al. 2023. Pond: Cxl-based memory pooling systems for cloud platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*. 574–587.

[47] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. 2000. Architectural support for copy and tamper resistant software. *Acm Sigplan Notices* 35, 11 (2000), 168–177.

[48] David Lie, Chandramohan A Thekkath, and Mark Horowitz. 2003. Implementing an untrusted operating system on trusted hardware. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*. 178–192.

[49] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, et al. 2018. Meltdown: Reading kernel memory from user space. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 973–990.

[50] Sihang Liu, Korakit Seemakhupt, Gennady Pekhimenko, Aasheesh Kolli, and Samira Khan. 2019. Janus: Optimizing memory and storage support for non-volatile memory systems. In *Proceedings of the 46th International Symposium on Computer Architecture*. 143–156.

[51] David McGrew and John Viega. 2004. The Galois/counter mode of operation (GCM). *submission to NIST Modes of Operation Process* 20 (2004), 0278–0070.

[52] Seonjin Na, Jungwoo Kim, Sunho Lee, and Jaehyuk Huh. 2024. Supporting secure multi-gpu computing with dynamic and batched metadata management. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 204–217.

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

[54] Lifeng Nai, Yinglong Xia, Ilie G Tanase, Hyesoon Kim, and Ching-Yung Lin. 2015. GraphBIG: understanding graph computing in the context of industrial solutions. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[55] nextplatform.com. 2024. A coherent interconnect strategy: CXL absorbs Gen-Z. https://www.nextplatform.com/2021/11/23/finally-a-coherent-interconnect-strategy-cxl-absorbs-gen-z/. Online; accessed Jun, 2024.

[56] nextplatform.com. 2024. CXL and Gen-Z Iron Out a Coherent Interconnect Strategy. https://www.nextplatform.com/2020/04/03/cxl-and-gen-z-iron-out-a-coherent-interconnect-strategy/. Online; accessed Jun, 2024.

[57] nextplatform.com. 2024. PCI-Express 5.0: The unintended but formidable datacenter interconnect. https://www.nextplatform.com/2021/02/03/pci-express-5-0-the-unintended-but-formidable-datacenter-interconnect//. Online; accessed Jun, 2024.

[58] PCI-SIG. 2024. TEE Device Interface Security Protocol (TDISP). https://pcisig.com/tee-device-interface-security-protocol-tdisp. Online; accessed Nov, 2024.

[59] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 264–278.

[60] 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 *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*. IEEE, 183–196.

[61] Brian Rogers, Milos Prvulovic, and Yan Solihin. 2006. Efficient data protection for distributed shared memory multiprocessors. In *Proceedings of the 15th international conference on Parallel architectures and compilation techniques*. 84–94.

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

[63] Gururaj Saileshwar, Prashant J. Nair, Prakash Ramrakhyani, Wendy Elsasser, José A. Joao, and Moinuddin K. Qureshi. 2018. Morphable Counters: Enabling Compact Integrity Trees For Low-Overhead Secure Memories. *2018 51st Annual*

IEEE/ACM International Symposium on Microarchitecture (MICRO) (2018), 416–427. https://api.semanticscholar.org/CorpusID:52222143

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

[65] Samsung. 2024. Samsung CXL Solutions – CMM-H. https://semiconductor.samsung.com/us/news-events/tech-blog/samsung-cxl-solutions-cmm-h/. Online; accessed Jun, 2024.

[66] Samsung. 2024. Samsung Unveils Industry-First Memory Module Incorporating New CXL Interconnect Standard. https://news.samsung.com/global/samsung-unveils-industry-first-memory-module-incorporating-new-cxl-interconnect-standard. Online; accessed Jun, 2024.

[67] Rohit Sehgal, Vishal Tanna, Vinicius Petrucci, and Anil Godbole. 2024. Optimizing System Memory Bandwidth with Micron CXL Memory Expansion Modules on Intel Xeon 6 Processors. *arXiv preprint arXiv:2412.12491* (2024).

[68] Debendra Das Sharma. 2023. Compute Express Link (CXL): Enabling Heterogeneous Data-Centric Computing With Heterogeneous Memory Hierarchy. *IEEE Micro* 43, 2 (2023), 99–109. https://doi.org/10.1109/MM.2022.3228561

[69] Weidong Shi, H-h S Lee, Mrinmoy Ghosh, Chenghuai Lu, and Alexandra Boldyreva. 2005. High efficiency counter mode security architecture via prediction and precomputation. In *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 14–24.

[70] SNIA. 2020. Compute Express Link™ (CXL™): Memory and Cache Protocols. https://snia.org/sites/default/files/SDC/2020/130-Blankenship-CXL-1.1-Protocol-Extensions.pdf. Online; accessed Oct, 2024.

[71] Samuel W Stark, A Theodore Markettos, and Simon W Moore. 2023. How Flexible is CXL's Memory Protection? Replacing a sledgehammer with a scalpel. *Queue* 21, 3 (2023), 54–64.

[72] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. 2003. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *ACM International Conference on Supercomputing 25th Anniversary Volume*. 357–368.

[73] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, et al. 2023. Demystifying cxl memory with genuine cxl-ready systems and devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*. 105–121.

[74] Michael Szydlo. 2004. Merkle tree traversal in log space and time. In *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*. Springer, 541–554.

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

[76] Samuel Thomas, Kidus Workneh, Jac McCarty, Joseph Izraelevitz, Tamara Lehman, and R Iris Bahar. 2024. A Midsummer Night's Tree: Efficient and High Performance Secure SCM. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*. 22–37.

[77] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. 2008. *CACTI 5.1*. Technical Report. Technical Report HPL-2008-20, HP Labs.

[78] Chia-Che Tsai, Donald E Porter, and Mona Vij. 2017. {Graphene-SGX}: A Practical Library {OS} for Unmodified Applications on {SGX}. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. 645–658.

[79] Chia-Che Tsai, Jeongseok Son, Bhushan Jain, John McAvey, Raluca Ada Popa, and Donald E Porter. 2020. Civet: An efficient java partitioning framework for hardware enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*. 505–522.

[80] Muhammad Umar, Weizhe Hua, Zhiru Zhang, and G Edward Suh. 2022. Softvn: Efficient memory protection via software-provided version numbers. In *Proceedings of the 49th Annual International Symposium on Computer Architecture*. 160–172.

[81] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lippi, Marina Minkin, Daniel Genkin, Yuval Yarom, Berk Sunar, Daniel Gruss, and Frank Piessens. 2020. LVI: Hijacking transient execution through microarchitectural load value injection. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 54–72.

[82] Xin Wang, Jagadish Kotra, Alex Jones, Wenjie Xiong, and Xun Jian. 2024. Counterlight Memory Encryption. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 724–738.

[83] Xin Wang, Daulet Talapkaliyev, Matthew Hicks, and Xun Jian. 2022. Self-reinforcing memoization for cryptography calculations in secure memory systems. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 678–692.

[84] Anthony D Wood and John A Stankovic. 2002. Denial of service in sensor networks. *computer* 35, 10 (2002), 54–62.

[85] Yubin Xia, Yutao Liu, and Haibo Chen. 2013. Architecture support for guest-transparent vm protection from untrusted hypervisor and physical attacks. In *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 246–257.

[86] Yuanchao Xu, James Pangia, Chencheng Ye, Yan Solihin, and Xipeng Shen. 2024. Data Enclave: A Data-Centric Trusted Execution Environment. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 218–232.

[87] Chenyu Yan, Daniel Englender, Milos Prvulovic, Brian Rogers, and Yan Solihin. 2006. Improving cost, performance, and security of memory encryption and authentication. *ACM SIGARCH Computer Architecture News* 34, 2 (2006), 179–190.