Decoupled-DSM: A Scalable Secure Trusted Execution Environment for Multi-Host CXL Disaggregated Shared Memory

Yuxin Yang
Computer Science and Engineering
University of California, Santa Cruz
Santa Cruz, CA, USA
yyang460@ucsc.edu

Yuanchao Xu
Computer Science and Engineering
University of California, Santa Cruz
Santa Cruz, CA, USA
yxu314@ucsc.edu

***Abstract*—Confidential cloud computing based on Trusted Execution Environments (TEEs) has gained widespread adoption for protecting virtual machine (VM) memory from privileged software. Meanwhile, Compute Express Link (CXL) enables disaggregated shared memory (CXL-DSM), allowing hardware-coherent, in-place multi-host data sharing. However, existing CPU-centric TEEs are incompatible with CXL-DSM: they enforce memory ownership at the host level, enabling undetected cross-host access, and rely on encrypt-copy mechanisms that negate CXL-DSM performance benefits. This paper proposes Decoupled-DSM TEE, a scalable architecture for secure multi-host CXL-DSM. Decoupled-DSM retains memory allocation at the CXL device while offloading frequent permission checks to host-side TEEs, and introduces a host-CXL validation protocol that reuses reverse map table mechanisms to prevent forged mappings. Evaluation shows that Decoupled-DSM incurs only 7.3%–9.2% overhead over an insecure baseline.**

***Index Terms*—Compute Express Link (CXL), trusted execution environments (TEEs), memory access control**

I. Introduction

Confidential cloud computing has gained widespread adoption [1]–[3], driven by hardware-based Trusted Execution Environments (TEEs) such as AMD SEV-SNP [4], Intel TDX [5], and ARM CCA [6]. These TEEs enable users to protect sensitive code and data even from privileged software, including cloud providers, with a small trusted computing base. Major cloud platforms, including Amazon AWS [1], Microsoft Azure [2], and Google Cloud [3], have already deployed confidential computing services. These CPU TEEs protect confidentiality by transparently encrypting confidential Virtual Machine (VM) memory and enforce integrity and isolation through hardware managed metadata that validates memory ownership and access permissions, preventing a malicious hypervisor or another confidential VM from accessing the data of a confidential VM.

Driven by the growing demand for larger, cost-efficient memory capacity and bandwidth, Compute Express Link (CXL) has emerged as a key building block for next-generation cloud systems. As a low-latency, cache-coherent interconnect, CXL enables CXL-enabled Disaggregated Shared Memory (CXL-DSM), where multiple hosts concurrently share memory devices. This architecture increases effective memory capacity and bandwidth while enabling hardware-coherent, in-place data sharing. By eliminating memory duplication and costly data movement in traditional RDMA-based systems [7], [8], CXL-DSM significantly improves performance for data-intensive applications [8]–[12].

However, existing TEEs are unable to securely and efficiently support emerging CXL-DSM server architectures. Prior TEE designs, including enclave-based systems [13] and confidential VMs such as Intel TDX [5] and AMD SEV [14], are architected around a single CPU socket and its directly attached local memory. These designs fundamentally assume that TEE memory is exclusively owned, managed, and accessed by the corresponding processor. Based on this assumption, VM TEEs enforce isolation and access control through host-local page-level permission management and protect memory using hardware-supported encryption mechanisms.

These assumptions no longer hold in CXL-DSM systems, where memory is disaggregated and accessed by multiple hosts. Existing VM TEEs enforce memory ownership and access control strictly at the host level, without cross-host coordination or validation. As a result, a compromised OS or hypervisor on one host can create forged page table entries (PTEs) that map to CXL memory pages belonging to confidential VMs on another host, enabling unauthorized access without triggering isolation violations. Such cross-host accesses occur within the standard TEE threat model and remain undetected by existing TEE mechanisms, which continue to report a secure execution environment despite the breach (Section III-A).

Furthermore, this architectural mismatch prevents existing VM TEEs from supporting a key capability of CXL-DSM: **coherent, in-place multi-host data sharing**. To share data across hosts, existing TEEs rely on secure communication channels, such as network transfers or untrusted shared memory, which require software-based encryption, integrity protection, and explicit data copying. These mechanisms negate the performance benefits of CXL-DSM by introducing substantial

copying and serialization overheads and eliminating hardware-supported coherence, resulting in performance comparable to conventional server architectures without CXL-DSM. In an 8-host configuration, this overhead from using existing TEEs slows execution by up to 6.99× compared to an insecure CXL-DSM baseline (Section VIII).

A straightforward approach to enable secure multi-host sharing in CXL-DSM is to introduce a **Centralized-DSM TEE** at the CXL memory device. In this design, the CXL-side TEE serves as a trusted global authority that manages memory allocation, deallocation, sharing, and access permissions for all hosts. Each CXL memory page and its associated permissions are maintained in a centralized permission table, and every memory access from any host is validated against this centralized state. This design provides strong cross-host isolation and secure in-place sharing. However, because permission checks are performed on the critical path of every memory access, centralized enforcement becomes a scalability bottleneck. As the number of hosts and cores increases, the aggregate memory access rate grows rapidly, resulting in increased access latency and limited throughput, making this approach unsuitable for large-scale CXL-DSM deployments.

To address this limitation, we propose **Decoupled-DSM TEE**. The key idea is to retain secure memory allocation and sharing control at the CXL memory device while offloading frequent memory access permission checks to reuse existing host-side CPU TEEs. Since allocation and sharing operations are far less frequent than memory accesses, this decoupled design improves scalability by enabling local enforcement on each host while maintaining centralized control over cross-host memory ownership. However, naively decoupling access checks introduces new security vulnerabilities, as a compromised OS or hypervisor could create forged PTEs that map to unauthorized CXL memory pages. We identify that the root cause is the lack of host-CXL validation for host-local PTE legitimacy. To address this issue, we design a **host-CXL validation protocol** that reuses the Reverse Map Table (RMT) in existing CPU-side TEEs to enforce authorized CXL memory access locally, ensuring secure and scalable enforcement.

We implement and evaluate Decoupled-DSM TEE on data-intensive workloads. The results show that Decoupled-DSM provides strong security guarantees with low overhead. Compared with state-of-the-art TEE-based sharing approaches [15] and Centralized-DSM TEE, Decoupled-DSM achieves substantially better performance while enabling secure, coherent, multi-host in-place sharing.

Overall, our main contributions are as follows:

- **We identify a security and performance gap** between CPU-centric TEEs and CXL-DSM, showing that existing TEEs fail to provide secure cross-host access control and sharing.
- **We propose Decoupled-DSM TEE**, which separates CXL-side allocation/sharing management from host-side access enforcement through host-CXL validation.

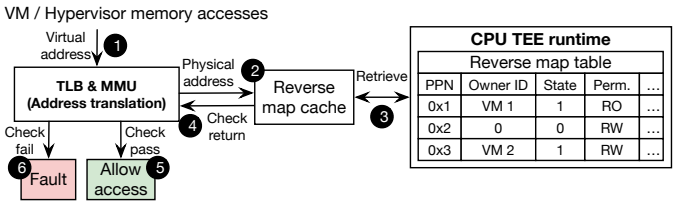


Fig. 1: A high-level workflow of VM TEE memory accesses.

- **Evaluation.** Across 1–8 hosts, Decoupled-DSM incurs only 7.3%–9.2% overhead over insecure CXL-DSM, reduces security overhead by 41.2% over Centralized-DSM.

II. Background

A. CXL Disaggregated Shared Memory

CXL provides a low-latency, cache-coherent interconnect for processors, accelerators, and memory devices [16]–[19]. It defines CXL.io for device discovery/configuration, CXL.cache for device participation in host coherence, and CXL.mem for host load/store access to device-attached memory. In CXL memory expansion, device memory is exposed as a host physical address range, often as a remote NUMA node; cache misses to this range are issued as CXL.mem read/write requests [20].

CXL memory pooling allows device-attached memory to be flexibly assigned to multiple hosts. A CXL Fabric Manager configures the fabric, partitions memory capacity, and programs address visibility [11]. CXL-DSM further allows multiple hosts to map the same CXL pages and access them in place through normal load/store instructions with coherent cache-line sharing, avoiding the duplication and data movement of traditional RDMA-based systems [8].

B. Trusted Execution Environments (TEEs)

Confidential cloud computing uses VM TEEs, such as Intel TDX [5] and AMD SEV-SNP [4], to protect guest memory from compromised system software. These CPU-centric TEEs encrypt guest memory with hardware-managed keys and enforce integrity/isolation using metadata such as reverse page tables, which validate page ownership and permissions. As shown in Figure 1, address translation triggers a reverse-map-cache check ❶–❷; on a TLB miss or LLC miss, the processor consults the reverse map table to confirm the ownership and permission of this permission ❸, and the result determines whether the access is allowed ❹–❻.

C. CXL Integrity and Data Encryption (IDE)

To secure data in transit, the CXL 2.0 specification introduces Integrity and Data Encryption (IDE) [21], which establishes secure communication between CXL ports. CXL IDE uses 256-bit AES in Galois/Counter Mode (AES-GCM) [22] to provide confidentiality, integrity, message uniqueness, and replay protection for data transmitted over the physical link. IDE operates at the per-link level: each host connected through a

different CXL link maintains a separate encryption tuple, such as key and initialization-vector state, to protect its link-level traffic independently. However, standard IDE only protects the communication channel against wiretapping and physical probing. It does not provide fine-grained, execution-context-aware access control or cross-host permission management for shared memory regions.

III. Motivation

A. New Security Gaps of TEEs in CXL-DSM

TEEs [4], [5], [14], [23] assume a tight hardware trust boundary where computation, memory controllers, and DRAM are co-located within one CPU package. CXL-DSM breaks this assumption by exposing remote, device-managed memory that can be shared and accessed by multiple hosts. As a result, memory is no longer exclusively controlled by one processor, and accesses may bypass CPU-centric protections designed for local memory.

We present a proof-of-concept attack showing that unauthorized CXL-DSM accesses can bypass existing VM TEE protection and detection. The attack follows the standard VM TEE threat model [4], [5]. We use AMD SEV-SNP [4] as an example, but the vulnerability broadly applies to confidential VM TEEs.

Attack Preconditions and System Assumptions. Before describing the attack, we assume the following system configuration and deployment conditions, as shown in Figure 2.

- **Multi-Host CXL-DSM Configuration.** A CXL memory module is configured by the CXL Fabric Manager to be visible and accessible to multiple hosts to support flexible workload migration and memory sharing, which is common in existing CXL-DSM systems [8]–[10]. Each host is equipped with a TEE-capable processor and operates within a correctly initialized TEE. The TEE processor on each host independently maintains hardware-managed reverse page tables to track ownership and access permissions of both local memory pages and CXL memory pages allocated by that host.
- **Victim Confidential VM Deployment with CXL Memory.** On Host 1, a victim confidential VM (VM A) is successfully launched with valid attestation under AMD SEV-SNP and allocates memory backed by the shared CXL memory module. The allocated CXL memory pages are encrypted using the VM's memory encryption key and are recorded and protected by Host 1's hardware-managed reverse page table.
- **CXL Link Protection.** CXL Integrity and Data Encryption (IDE) provides link-level confidentiality and integrity protection for data transmitted over the CXL links [21].

Attack Steps and Why They Bypass Existing TEE Enforcement. The following steps illustrate a proof-of-concept attack that bypasses existing VM TEE protections.

- **Step 1:** The compromised OS or hypervisor on Host 2 constructs a forged Page Table Entry (PTE) that maps its Virtual Address (VA) to CXL physical page 2, which has already been allocated to the victim VM A on Host 1. This mapping succeeds on Host 2 without triggering ownership or permission violations in Host 2's hardware reverse page table, because none of Host 2's existing VMs have previously claimed ownership of this CXL physical page. This occurs because memory ownership is enforced only at the local host level, and no cross-host ownership validation is performed for CXL physical pages, as shown in Figure 2 ❶.
- **Step 2:** The compromised OS or hypervisor on Host 2 issues regular load and store instructions to the maliciously mapped CXL page 2. Because the CXL memory module is accessible to Host 2 at the fabric level, these memory requests are routed through the CXL fabric to the shared CXL memory device and are serviced successfully, resulting in unauthorized access to the victim VM A's CXL memory, as shown in Figure 2 ❷.
- **Step 3:** The compromised OS or hypervisor repeats Steps 1 and 2 to perform arbitrary unauthorized accesses to Memory Encryption Engine (MEE) encrypted CXL memory pages belonging to the victim VM A or other VMs.

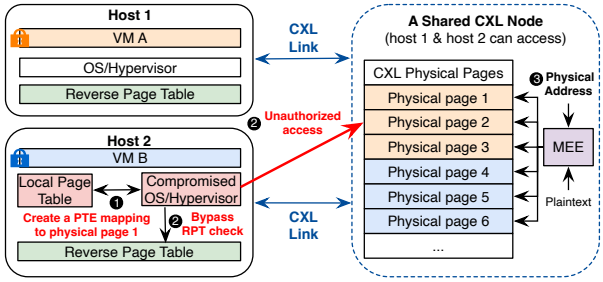


Fig. 2: A cross-host attack using a fake PTE.

Throughout the attack, existing VM TEE mechanisms on both hosts report no allocation or memory ownership violations, nor unauthorized accesses, even as the compromised OS or hypervisor allocates and accesses CXL memory pages belonging to a guest VM on another host.

Takeaway 1. Existing VM TEEs are insecure in CXL-DSM environments because their assumptions of local, exclusive, single-CPU-controlled memory no longer hold. The lack of cross-host ownership validation at allocation time and access-time permission enforcement allows unauthorized accesses to bypass existing VM TEE protection and detection.

B. TEE Performance Limits on CXL-DSM

*1) **Qualitative Analysis**:* CXL-DSM improves performance and resource efficiency by decoupling memory from individual servers and enabling coherent, in-place multi-host data sharing [8]–[10]. Figure 3(a) illustrates an example, Tigon, demonstrating how CXL-DSM can improve database performance [10]. CXL-DSM provides two key benefits. **(1) Multi-host coherent in-place sharing:** compute nodes can directly access a shared database using native load/store instructions, reducing data-sharing latency and coherence overhead compared with conventional network-based techniques such as

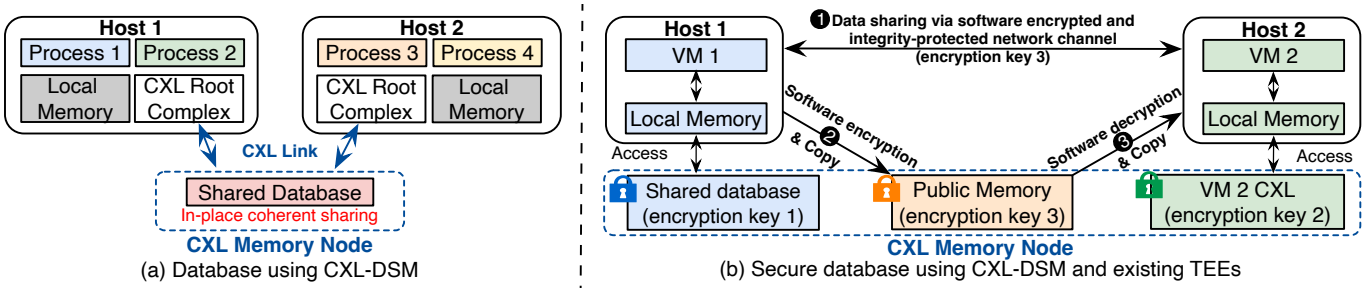


Fig. 3: Examples of a CXL-DSM-based database and a secure CXL-DSM database using existing TEEs.

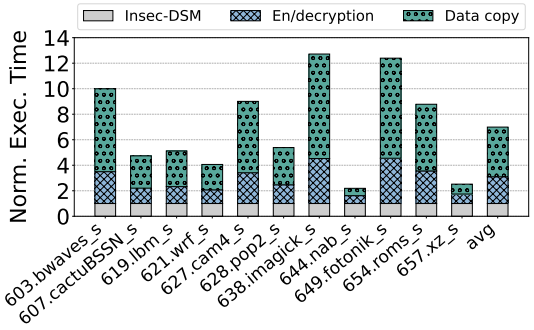


Fig. 4: Overhead breakdown of Encrypt-Copy-DSM.

RDMA [24]. **(2) Scalable resource allocation:** additional compute nodes can attach to the shared memory pool without incurring the overhead of transferring database contents over the network. By combining these properties, Tigon [10] demonstrates that CXL-DSM improves database throughput by an average of 2.5× compared with traditional servers provisioned with the same compute and memory resources but without CXL-DSM.

Figure 3(b) illustrates the performance limitations of existing TEEs in CXL-DSM systems, which do not support secure multi-host in-place sharing. Under current TEE-based sharing models, VMs on different hosts must communicate through an untrusted channel even after mutual attestation [15], [25]. As shown in the figure, VM 2 cannot directly access VM 1's shared database. Instead, the VMs first establish a secure channel by exchanging encryption keys ❶. VM 1 then encrypts the data, computes a Message Authentication Code (MAC), and copies the ciphertext into a public CXL memory region ❷. VM 2 subsequently retrieves the data, decrypts and verifies it, and places it into its own memory ❸. This path of software encryption and copy eliminates the in-place sharing advantage of CXL-DSM. Moreover, coherence is no longer preserved because copied data reside at different PAs. Consequently, VMs must either explicitly track coherence and exchange invalidation or update messages, or fall back to one-time sharing without retaining destination-side copies.

*2) **Quantitative Analysis**:* As analyzed in the previous section, naively extending existing TEEs to CXL-DSM systems introduces substantial encryption and data-copy overhead. To quantify the overhead of the software-based encryption and copy approach (Encrypt-Copy-DSM) used by existing TEEs relative to an insecure CXL-DSM baseline without TEE protection (Insec-DSM), we conduct experiments using ChampSim [26] under memory-intensive workloads, where all application memory is allocated in CXL memory. We evaluate configurations with 1 to 8 hosts, each with two cores, and compare the end-to-end latency of an insecure CXL-DSM system with that of a software-based Encrypt-Copy CXL-DSM system. Detailed evaluation methodology is in Section VIII. The results in Figure 4 show that encryption and data copying increase end-to-end latency by 6.99× relative to an insecure CXL-DSM system in an 8-host configuration. The software-based en/decryption is 35.1% of the extra overhead Encrypt-Copy-DSM introduces.

Takeaway 2. Existing software-based TEE extensions for CXL-DSM suffer from substantial encryption/decryption and data-copy overheads, especially as the number of hosts increases. They lack an efficient and scalable architecture for minimizing security overhead and mitigating performance degradation at scale.

C. Limitations of TEE-Based Secure Sharing

Prior studies have explored secure memory sharing under TEEs within a single-CPU environment [15], [25], [27]. These approaches enable secure data sharing among enclaves or confidential VMs by directly mapping memory between VMs/enclaves running on the same CPU, eliminating software-based encryption and data copying. However, these designs rely on a single trusted CPU to fully manage memory mappings and access permissions, assuming that shared memory remains physically local and exclusively controlled by one processor. Consequently, their security and performance models do not extend to CXL-DSM, where memory is disaggregated, remote, and accessible by multiple hosts without a single global trusted controller. Although TEE-related support for CXL-DSM appears in the CXL 3.0, 3.1, and 4.0 specifications [28]–[30], these specifications do not provide a concrete TEE architecture for secure, efficient, and scalable multi-host CXL-DSM systems.

IV. Threat Model

Our threat model follows VM TEEs as prior work [4], [5] as follows.

Adversary. We consider an adversary capable of compromising the OS or hypervisor on one or multiple hosts and exploiting exposed system interfaces, including runtime system calls, memory access mechanisms, and OS-level capabilities such as page table creation and management. The adversary may create attacker-controlled VMs on one or multiple hosts and issue exposed system calls and arbitrary memory access requests through these VMs. In addition, the adversary can observe, modify, and replay data transmitted over the CXL links. The adversary may further combine any of these capabilities to launch coordinated attacks.

Assumption. We assume that the TEE runtime operates in good faith. Specifically, the runtime is assumed to correctly execute benign instructions as intended and not introduce malicious or incorrect behavior. Following prior work [25], [27], we assume that data sharing between VMs occurs only among attested and mutually trusted VMs. VMs do not intentionally share data with malicious or unknown VMs.

Out of Scope: We do not defend against denial-of-service attacks [31], physical attacks on trusted hardware [32], compromise of the underlying cryptographic primitives [33], or side-channel attacks [34] unrelated to the Decoupled-DSM permission-enforcement path.

V. Design Overview

A. Design Goals

To address the performance and security limitations of existing TEEs on CXL-DSM, our design pursues the following Goals (G):

- **G1: Secure multi-host in-place sharing:** Enable secure multi-host in-place sharing using native load/store semantics while eliminating the overheads of encryption and data copying.
- **G2: Secure CXL-DSM page allocation and access:** Provide hardware-enforced cross-host CXL-DSM page allocation and access control to ensure that only authorized tenants can allocate and access shared CXL memory pages.
- **G3: Scalable multi-host security enforcement:** Support scalable security enforcement across multiple hosts, avoiding performance bottlenecks as the number of hosts increases.

B. A Strawman Design: Centralized-DSM TEE

Existing TEEs lack a centralized, cross-host unit to manage CXL-memory allocation and access permissions in CXL-DSM. To address this root cause, we propose **Centralized-DSM TEE**, which enforces allocation/deallocation, isolation, and permission checks directly on the CXL memory device, as shown in Figure 5.

Centralized-DSM TEE provides memory-sharing and management primitives to confidential VMs across multiple hosts through system calls. These calls are forwarded to the centralized Centralized-DSM TEE runtime, which validates requests to prevent unauthorized or inconsistent allocation/deallocation. For each CXL-memory page, the runtime records VM access permissions in a **Permission Table**, satisfying **G1**. To

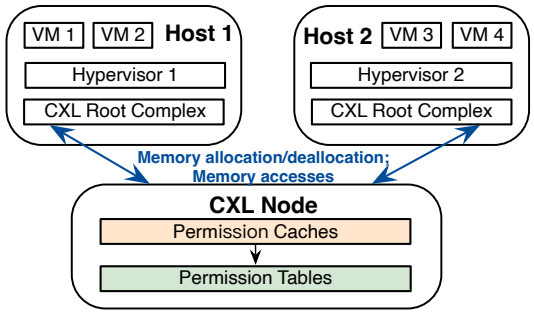


Fig. 5: Centralized-DSM architecture overview.

enforce permissions efficiently at access time, the CXL device maintains hardware **Permission Caches** that store active permission-table entries, satisfying **G2**. Centralized-DSM is independent of host-side TEEs; existing CPU TEEs only need lightweight libraries that expose these system calls to confidential VMs, improving compatibility and deployment ease.

Scalability Limitation. Although Centralized-DSM TEE is conceptually simple, it does not scale to CXL-DSM systems, where multiple hosts concurrently access shared memory. Since every memory access must be validated against centralized permission metadata, permission checks remain on the critical path. As hosts and cores increase, the aggregate access rate makes centralized enforcement a bottleneck, increasing latency and limiting throughput, thus failing to satisfy **G3**. Permission caches can reduce access-time checks, but they must serve requests from all cores across all hosts, effectively aggregating system-wide TLB demand. High hit rates require large caches, causing high hardware cost and longer lookup latency, while smaller caches incur frequent misses and permission-table accesses. Thus, Centralized-DSM TEE still incurs substantial overhead and scales poorly with more hosts and cores.

C. Decoupled-DSM TEE Overview

Design Initiation. To preserve **G1** and **G2**, which are satisfied by Centralized-DSM TEE, while improving scalability to meet **G3**, we propose **Decoupled-DSM TEE**. The key idea is to decouple memory access permission checks from the CXL memory node and reuse existing CPU-side TEEs to securely enforce CXL memory accesses on each host to improve scalability. Since permission checks for memory accesses occur far more frequently than memory allocation, deallocation, and sharing primitives, Decoupled-DSM TEE performs memory access permission checks at the CPU-side TEE while retaining the execution of memory allocation, deallocation, and sharing primitives at the Decoupled-DSM TEE on the CXL memory node. This separation improves scalability while preserving secure allocation, sharing, and access permission checks.

However, naively adopting this design introduces new security vulnerabilities, as illustrated in Section III-A. A compromised OS or hypervisor could create forged local PTEs that directly map to CXL physical pages without authorization

from the CXL memory node. These forged PTEs may bypass local CPU-side TEE permission checks, leading to unauthorized access.

We identify that the root cause is the lack of a host-CXL validation mechanism to verify host-local PTE legitimacy. To address this issue, we design a **Host-CXL Validation Protocol** that reuses the Reverse Map Table (RMT) in existing CPU-side TEEs as the local ground truth for CXL memory allocation, deallocation, and permissions. In our design, RMT entries for CXL memory can only be created or updated with authorization from the Decoupled-DSM TEE on the CXL memory node. Consequently, even if a compromised OS or hypervisor creates forged PTEs without approval, subsequent accesses using these PTEs will fail RMT validation and be detected locally on the host, ensuring secure and scalable enforcement.

Architectural Design Overview. Figure 6 shows the overview of Decoupled-DSM consisting of four key components: (1) Decoupled-DSM TEE runtime on CXL memory side, (2) the permission table cache on CXL memory side, (3) a modified CPU TEE runtime with Decoupled-DSM TEE library with Local CXL Reverse Map Table (LCRMT), and (4) a local CXL reverse map cache that stores frequently used LCRMT entries. Decoupled-DSM supports memory in-place sharing across multi-host through a set of system calls handled by the single trusted component Decoupled-DSM TEE runtime and permission table cache on CXL node, satisfying **G1** and **G2**. On each host, the modified CPU-side TEE runtime performs permission checks locally using LCRMT for every memory access. The LCRMT can be updated only through Decoupled-DSM system calls and validated via the host-CXL validation protocol, satisfying **G1** and **G3**.

Trust Computing Base (TCB) of Decoupled-DSM. The TCB of Decoupled-DSM TEE follows the trust assumptions of AMD SEV-SNP [4] and CXL Integrity and Data Encryption (IDE) [21] defined in the CXL specification. Under this model, TCB includes the CPU processor, CPU-side TEE runtime, CXL root complexes on both the host and the CXL memory module, the CXL controller, and the CXL memory controller. Decoupled-DSM preserves the existing trust assumptions and adds new components in existing TCB, including the Decoupled-DSM TEE library and the local CXL reverse map table within the CPU-side TEE runtime, as well as the permission table cache on the CXL controller. In addition, Decoupled-DSM introduces a new TCB, the Decoupled-DSM TEE runtime on the CXL memory module.

VI. Detailed Design of Decoupled-DSM TEE

A. Memory Management System Calls.

Existing CXL-DSM systems adopt coarse-grained programming models for shared memory management [35]–[37], typically providing sharing semantics at the user, group, and other levels similar to Unix file system access control [38]. Under this model, VMs or processes with the same user or group identity share identical access permissions, which are managed by the host OS. However, this approach is incompatible with

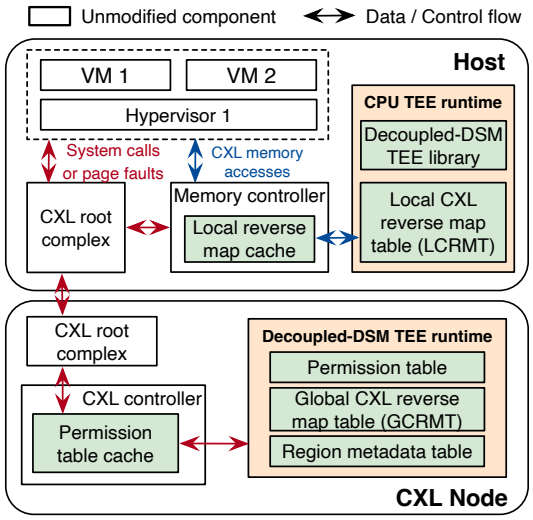


Fig. 6: Decoupled-DSM architecture overview.

the TEE threat model, where the OS is excluded from the TCB. Relying on OS-managed identities and permissions for shared CXL memory therefore introduces security risks, as a compromised OS can manipulate identity assignments or permission metadata to bypass access control [39].

Existing VM TEEs only support copy-based, point-to-point data sharing between two confidential VMs. Such sharing happens only after successful VM mutual attestation and a secure communication channel is established between two VMs. This method has two limitations: (1) Point-to-point sharing does not scale naturally beyond two VMs. When multiple confidential VMs need to share the same data, the system must create multiple independent copies and communication channels, increasing complexity and overhead. (2) Existing VM TEE sharing mechanisms do not support fine-grained read/write/execute (R/W/X) permission controls. Once a VM receives copied data from another VM, the data owner loses control of how the recipient VM subsequently accesses or shares that data.

TABLE I: Decoupled-DSM system calls.

System calls	Details	Permitted Conditions
create(size)	Create a shared memory region on CXL memory; return RID.	Available CXL memory space.
destroy(RID)	destroy a shared memory region RID on CXL memory.	Caller is the owner VM of this region.
grant(VMID,P,RID)	grant permission P to VM VMID for region RID.	Caller is the owner VM of this region.
revoke(VMID,P,RID)	revoke permission P from VM VMID for region RID.	Caller is the owner VM of this region.
mmap(addr,RID,P)	map the region RID at virtual address addr in caller VM with permission P.	Caller is the owner VM or sharee VM of this region RID; the requested permission P is granted before.
munmap(addr,RID)	unmap region RID at virtual address addr in caller VM.	Caller is the owner VM or sharee VM of this region RID.

To align with the TEE threat model while enabling coherent, in-place multi-host data sharing, we propose a new set of Decoupled-DSM TEE sharing primitives, as shown in Table I. These primitives provide the following four features.

Decoupled management and access permissions separate management permissions from access permissions. Manage-

ment permissions define ownership of a shared memory region and authorize a VM to grant or revoke access permissions to other VMs, while access permissions specify how a shared memory region may be accessed, including read, write, and execute permissions.

Coherent in-place multi-host sharing allows VMs to create and delete shared memory regions and manage sharing across hosts using per-VM access permissions. By enforcing per-VM read, write, and execute permissions, Decoupled-DSM TEE enables coherent, in-place sharing of CXL memory across hosts, eliminating data copying while preserving isolation.

Dynamic and asymmetric access permissions enable different VMs to hold distinct access permissions for the same shared memory region. Permissions can be dynamically adjusted to match application requirements while maintaining isolation. For example, in a producer–consumer pattern, the producer VM may have write permission, while the consumer VM is restricted to read-only.

Explicit secure point-to-point permission management retains the point-to-point sharing model of VM TEEs, where access permissions are explicitly granted to or revoked from a specific VM only after completing the standard TEE authorization flow.

Table I summarizes the Decoupled-DSM TEE system calls for managing CXL memory regions. A confidential VM creates a region with a unique region ID (RID) using `create`; only its owner may `destroy` it or `grant` permissions to other VMs. The owner and authorized VMs may `mmap` and `munmap` the region subject to granted permissions. These calls are exposed to confidential VMs within host-side TEEs.

B. Decoupled-DSM TEE Design

1) ***Data Structure on Decoupled-DSM TEE runtime and CPU TEE runtime.****:* To support Decoupled-DSM memory-sharing system calls, we design a Decoupled-DSM TEE runtime with control logic and metadata structures to track shared memory regions and enforce permissions. Specifically, the Decoupled-DSM TEE runtime maintains three tables on the CXL side for protected shared memory regions. The **Permission Table** stores access permissions for each region using 64-byte entries indexed by Region ID. Each entry contains a 10-bit Region ID, a PPN that points to the first page of this region's page table and 32 slots, each consisting of a 13-bit VM ID and 3-bit permission (read, write, execute), allowing a region to be shared among up to 32 VMs. The **Region Metadata Table** records region-level information, including a 10-bit Region ID, 13-bit owner VM ID, 36-bit region size, 36-bit page counter tracking allocated pages, and a 512-bit per-region encryption key. The **Global CXL Reverse Map Table (GCRMT)** tracks page ownership by mapping each physical page number (PPN) to its corresponding region. Each entry includes the PPN, a valid bit indicating whether the page has been allocated, and a 10-bit Region ID. The Decoupled-DSM TEE runtime also includes a memory allocator that manages page allocation and deallocation for region during `create()` and `destroy()` operations using the GCRMT.

The CPU-side TEE runtime maintains a Local CXL Reserved Mapping Table (LCRMT), which records only the CXL pages allocated to the corresponding host[1]. The LCRMT serves a role similar to the Reverse Mapping Table in AMD SEV-SNP. Instead of relying solely on PTEs, which may be forged by a malicious OS or hypervisor, each memory access is validated against the LCRMT to ensure legitimacy. We added the Local CXL Reverse Mapping Cache (LCRMC) to cache frequently accessed LCRMT entries, accelerating permission checks for memory accesses. The LCRMT is indexed by PPN. Each entry contains 1 valid bit and 16 slots, where each slot consists of a 13-bit VM ID and a 3-bit permission field (read, write, execute), allowing a page to be shared among up to 16 VMs.

2) ***Host-CXL Validation Protocol****:* Decoupled-DSM TEE performs permission management at region granularity, which aligns with existing memory-sharing semantics and simplifies access control for shared memory regions. However, memory allocation in CXL memory is performed at page granularity to reduce fragmentation and avoid requiring contiguous physical memory. Similarly, the CPU-side Local CXL Reserved Mapping Table (LCRMT) and its associated cache operate at page granularity. Consequently, the CPU-side CXL root complex not only forwards Decoupled-DSM system calls but also participates in page-level permission enforcement by forwarding page faults through a Host–CXL validation protocol.

Under this design, when a VM accesses a CXL memory page that does not have a valid PTE entry, a page fault is generated and forwarded by the CPU-side CXL root complex to the CXL module. The Decoupled-DSM TEE runtime then consults the region metadata table, permission table, and global reverse page table to validate whether the requesting VM has the required permissions for the corresponding region. If the access is authorized, the Decoupled-DSM TEE runtime allocates or identifies the corresponding physical page and returns the validated mapping to the requested host. Upon receiving the validation response, the CPU-side runtime installs the PTE in the requesting VM's page table and updates the LCRMT with the corresponding page-level permissions. Subsequent memory accesses to the same page are validated locally through the LCRMT and its cache, avoiding repeated remote validation and reducing access latency.

The Host–CXL validation protocol is the only mechanism allowed to update entries in the LCRMT, ensuring the legitimacy of PTEs for CXL memory pages. Specifically, LCRMT entries are installed only after successful validation through the CXL-side Decoupled-DSM TEE runtime during page fault handling. This design prevents unauthorized mappings from being established locally. If a malicious OS or hypervisor attempts to create a forged PTE that maps to a CXL memory page without going through the Host–CXL validation protocol, no corresponding entry will exist in the LCRMT. Consequently, when the access is issued, the CPU-side TEE

[1]For simplicity, we assume each host contains a single CPU.

runtime checks the LCRMT, detects the absence of a valid entry, and denies the access.

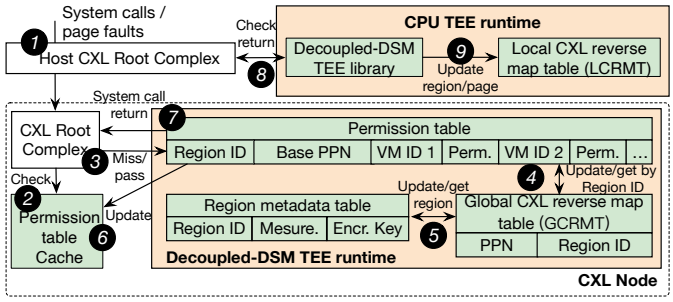


Fig. 7: Workflow of system calls and page faults.

*3) **Workflow of system calls and page fault handling**:* As illustrated in Figure 7, Decoupled-DSM handles system calls and page faults issued from hosts. When a host issues a system call or page fault, the request is transmitted to the CXL node through the CXL root complex using CXL IDE protection ❶. The request, along with its associated metadata, is encrypted in transit and decrypted and verified at the CXL root complex on the CXL node.

After verification, system calls are forwarded to the Decoupled-DSM TEE runtime ❸. The runtime verifies whether the caller is authorized to execute the requested operation by consulting the relevant metadata tables. If the verification succeeds, the runtime processes the request and updates the Global CXL Reverse Page Table if applicable ❹, the region metadata table if applicable ❺, and the permission table cache if applicable ❻, before returning a success signal ❼. If verification fails, an interrupt is raised as a violation.

For page faults, the request includes the region ID, requesting VM ID, and the offset within the region. The request is first checked against the permission table cache ❷. If permitted, the runtime retrieves the region base PPN, which points to the highest-level page table directory for the region, using the region ID, and performs a page table walk to compute the corresponding physical page based on the requested offset ❸. The validated mapping is then returned to the host CPU TEE runtime ❽, which updates the page table and the LCRMT.

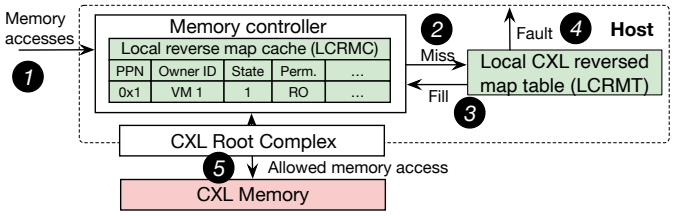


Fig. 8: Memory access workflow.

*4) **Workflow of Memory Accesses**:* Figure 8 illustrates the workflow for Decoupled-DSM protected memory access. Unlike system calls, permission checks for memory accesses are performed entirely on the host side. First, a CXL memory access is sent to the LCRMC in the memory controller for lookup ❶. On a cache miss ❷, the corresponding entry is fetched from the LCRMT and inserted into the LCRMC ❸. If the permission check succeeds, the request proceeds to access CXL memory through the CXL module ❺. Otherwise, if no valid entry exists in the LCRMT or the permissions do not authorize the access, a fault signal is raised to indicate an access violation ❹. Decoupled-DSM does not alter the CXL controller path for unprotected memory. Following existing VM TEEs, it uses reverse-map table/cache metadata to distinguish TEE-protected CXL accesses from unprotected accesses.

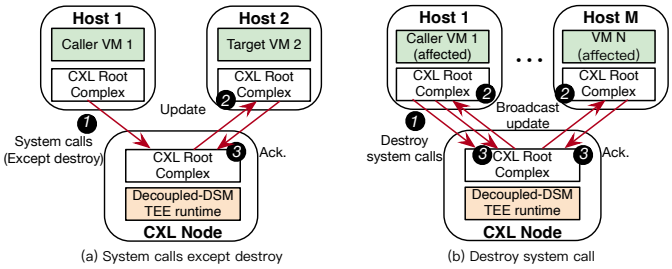


Fig. 9: Workflow of different system calls.

5) Workflow of System Calls.: Decoupled-DSM avoids distributed coherence or cross-host synchronization except for the destroy system call. This property comes from two design choices. First, each host maintains in its LCRMT only the up-to-date CXL-memory mappings of its own VMs, without caching or tracking mappings from other hosts. Second, all system calls except destroy modify permissions and mappings for only one host. Thus, they only update that host's page tables, TLBs, LCRMT, and the CXL-side metadata, without requiring cross-host coherence.

Decoupled-DSM also prevents stale mappings in other hosts' LCRMTs or TLBs through the Host-CXL Validation Protocol. Any cached CXL-memory mapping must correspond to an active permission entry validated by Decoupled-DSM TEE metadata. When a VM's permission to a CXL memory region changes due to grant or revoke system calls, the corresponding LCRMT and TLB entries are atomically invalidated before subsequent accesses. Even if a malicious OS creates forged PTEs, the access is detected during the page-table walk or memory access, since each TLB miss and LLC miss are validated against the trusted, up-to-date LCRMT, following the AMD SEV-SNP design (see Figure 8). The only exception is destroy, which revokes a shared CXL memory region and therefore must invalidate all related VMs' mappings across all related hosts before allowing subsequent operations.

Figure 9(a) shows the workflow for all system calls except destroy. The caller VM sends a system call, including the target VM's ID, to the CXL-side Decoupled-DSM runtime ❶. After verifying the caller's identity and permission, the runtime atomically updates the CXL-side metadata and the target VM's host-side metadata ❷, including the PTE, TLB, LCRMT, and LCRMC entries. The target host-side TEE returns an acknowledgment only after these updates complete and before processing subsequent memory accesses or system calls ❸.

Thus, later page faults and memory accesses observe the latest permission state on both the target host and the CXL-memory side. Note that the target VM may also be the caller VM or none for the create system call.

Destroy incurs extra overhead because it may affect multiple hosts. As shown in Figure 9(b), the caller VM first sends the destroy system call to the CXL-side Decoupled-DSM runtime ❶. After verifying the caller's identity and permission, the runtime blocks new management operations for the target region and sends invalidation requests to all affected hosts' VMs ❷. Each host-side TEE invalidates the corresponding PTE, TLB, LCRMT, and LCRMC entries for corresponding VMs and returns an acknowledgment ❸. The CXL-side runtime commits destroy only after receiving all acknowledgments, ensuring that no stale host-side metadata can authorize later accesses. Any subsequent access through an old PTE either misses in LCRMT/LCRMC or fails the updated permission check and raises a fault.

6) Workflow of CXL IDE and Device-side Encryption.: Figure 10 shows how Decoupled-DSM combines existing CXL IDE with device-side memory encryption. Decoupled-DSM separates data-in-transit protection from data-on-device protection: CXL IDE protects CXL flits on the host–device link, while the CXL memory controller encrypts data before storing them in CXL memory.

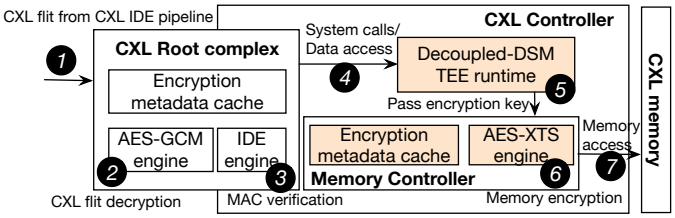


Fig. 10: Decoupled-DSM CXL IDE workflow.

For each system call or memory access, the host-side CXL root complex use the VM-specific IDE context to encrypt/authenticate the data flit ❶. The device-side IDE engine decrypts the flit ❷ and verifies its MAC ❸ before forwarding the request to the CXL controller. For writes to CXL memory, Decoupled-DSM performs encryption at the CXL memory controller. The Decoupled-DSM TEE runtime manages per-region encryption keys and passes the indexed encryption key to the memory encryption engine indexed by RID and VMID ❺. The memory controller then uses the encryption metadata cache and AES-XTS engine for memory en/decryption ❻.

C. Other Details

*1) **Per-VM CXL IDE Design.**:* Existing CXL IDE adopts a per-host protection model, where each host–device link is secured using a cryptographic tuple consisting of an encryption key, initialization vector, and synchronized message counter. While this design ensures link-level confidentiality and integrity, it does not prevent attacks in which a malicious VM or compromised OS on the same host impersonates another VM by issuing forged system calls or memory access requests.

To address this limitation, Decoupled-DSM TEE extends CXL IDE to a per-VM protection model, assigning each VM a unique cryptographic tuple for communication with the CXL memory module. The CPU-side CXL root complex verifies the identity of the sending VM before forwarding requests, preventing host-side VM impersonation. In addition, device-side cryptographic verification detects and rejects attempts to impersonate another VM through message modification over the CXL link. This design maintains independent cryptographic contexts per VM while leveraging the existing CXL IDE infrastructure.

*2) **Device-Side Encryption and Key Management.**:* To align with VM TEE security requirements, data stored in CXL memory must be encrypted. A key design choice is whether encryption is performed at the host or at the CXL memory device. Host-side encryption, as used in existing VM TEEs, requires sharing encryption keys across hosts for cross-host memory sharing, which is not securely supported and introduces additional attack surfaces.

Decoupled-DSM TEE therefore performs encryption at the CXL memory controller. Each shared memory region is protected using a per-region encryption key generated and managed by the Decoupled-DSM TEE runtime. Data transmitted over the CXL link is protected by CXL IDE for confidentiality and integrity in transit, while encryption and decryption of stored data occur entirely within the CXL memory module. This design enables secure multi-host sharing without cross-host key exchange and simplifies key management.

D. Security Analysis

We analyze Decoupled-DSM's security implications based on prior CPU VM TEE studies [4], [5], [40]. Under our threat model, adversaries may compromise system software, CXL links, malicious confidential VMs, or collude across them. Assuming correct implementation, Decoupled-DSM detects and rejects any system call or memory access that violates its permission model.

Unauthorized Access via Forged Host Page Table Entries. As discussed in Section III-A, an attacker may compromise the host OS to create forged PTEs that map to CXL pages owned by other VMs on other hosts, which existing VM TEEs do not prevent. Decoupled-DSM TEE prevents this attack through host–CXL validation of host-local PTE legitimacy using the host–CXL validation protocol and a local CXL reverse mapping table. Forged mappings created without invoking Decoupled-DSM TEE system calls cannot be inserted into this table. Thus, accesses through such mappings fail local validation, are rejected, and raise an interrupt.

Forged System Calls and Memory Accesses via VM Identity Spoofing. An attacker may spoof another VM's identity to issue forged system calls or memory accesses, either from a malicious VM, compromised OS, or by tampering with CXL messages in transit. Decoupled-DSM TEE mitigates these attacks using CXL IDE for per-VM protection and sender

verification. Host-side spoofing is blocked by the CPU-side CXL root complex, while in-transit tampering is detected and rejected by cryptographic verification at the CXL memory device.

Stale LCRMT Entries after Revocation or Destruction. Decoupled-DSM does not allow a VM to keep accessing a destroyed region through stale LCRMT/TLB entries. Unlike other system calls, destroy is the only operation that requires cross-host synchronization: before destroy commits, the CXL-side runtime blocks new management operations for the target region and sends invalidation requests to all affected hosts. Each host-side TEE invalidates the corresponding PTE, TLB, LCRMT, and LCRMC entries and returns an acknowledgment. The CXL-side runtime commits destroy only after receiving all acknowledgments.

E. Implementation Feasibility and Overhead

1) Hardware Overhead.: As shown in Table II, Decoupled-DSM adds a 4KB LCRMC on each host. On the CXL side, Decoupled-DSM adds a 4KB permission table cache, a 4KB encryption metadata cache, 4KB per-VM IDE contexts, and 64KB per-region key storage, totaling 76KB on-chip storage.

TABLE II: Area overhead for each host and CXL node.

Hardware	Configuration	# of entries	Overhead
Local reverse map cache	64 entries	64B per entry	4KB
Permission table cache	64 entries	64B per entry	4KB
Encryption metadata cache	4KB	1	4KB
Per-VM IDE contexts	64B per context	64	4KB
Per-region key	64B	1024	64KB
AES-XTS engine	1	8 pipelines	0.88mm^2 380.8mW

FPGA implementation for CXL controller to implement Decoupled-DSM is practical because current CXL memory platforms, such as Intel's CXL memory platform, already use FPGA logic to implement CXL controller functionality [20]. Decoupled-DSM only adds lightweight control logic in its design. For the evaluated 8-host setup, we estimate 10K–20K LUTs, 8K–18K flip-flops, and 25–35 BRAM36 blocks, excluding AES soft logic. AES can be provided by hardened AES/security IP or existing CXL IDE/security engines.

2) Device-side cryptographic power overhead.: Table II reports the hardware overhead of Decoupled-DSM. In the evaluated 8-host configuration, the total added on-chip storage is 108KB including all host-side LCRMCs. The AES-XTS datapath dominates the conservative upper-bound cost. To sustain the evaluated dual-channel DDR5-6400 memory bandwidth, Decoupled-DSM uses 8 line-rate AES-XTS pipelines, adding about 0.88mm^2 area and 380.8mW power in 45nm. Using CACTI [41], the 108KB added storage contributes about 96mW leakage power, leading to a conservative total power overhead of about 0.48W if the AES-XTS datapath is counted as entirely new hardware.

3) CXL-side runtime feasibility.: We expect future CXL controllers to incorporate increasingly sophisticated control-plane logic for security purposes, while still providing small computation capability compared with host CPUs. This trend is already reflected in the CXL specification: for example, CXL Link-level Integrity and Data Encryption, introduced in CXL 2.0 [42], requires additional encryption/integrity engines and key-management support on the CXL device side. Current commercially available CXL memory platforms, such as Intel's CXL memory platform, use FPGA to implement CXL controller logic [20], [43], suggesting that adding limited trusted control-plane functionality is feasible. This small device-side computation capability motivates Decoupled-DSM's decoupled design: the CXL-side trusted logic handles only low-frequency control operations, while frequent access-time permission checks are moved to the host-side CPU TEE.

VII. Evaluation Methodology

Simulator. We evaluate all schemes using ChampSim (version 2024.12) [26], a trace-driven microarchitectural simulator. Table III summarizes the simulation parameters. We model system-call cost using page-table update overheads 2us per call from prior work [44], plus the required CXL round trips, and include cross-host synchronization only for destroy.

TABLE III: Simulator parameters

Processor Configuration	
CPU	2 cores per host, OOO, x86-64, 4.00GHz
L1 TLB	8-way, 64 entries, Access latency: 1 cycles
L2 TLB	12-way, 1536 entries, Access latency: 8 cycles
L1I and L1D Cache	8-way, 32KB, 64B block, Access latency: 2 cycles
L2 Cache	16-way, 512KB, 64B block, Access latency: 20 cycles
L3 Cache	32-way, 4MB, 64B block, Access latency: 30 cycles
Memory and Memory Controller Configuration	
Per-channel CXL/local memory	DDR5 6400MHz, tRCD/tCL/tRP/tRAS/ tWR =14/14/14/32/30ns
Memory Channel	1 channel per 4 hosts for CXL memory 1 DRAM channel local memory for each host
System call latency	2us + CXL round trips; destroy adds cross-host sync.
En/decryption, MAC latency	4KB encryption metedata cache, 40 cycles per cacheline
Reverse map cache	Decoupled-DSM: 4KB per host, 30-cycle latency Centralized-DSM: aggregate cache size is 4KB × # of hosts
Permission table cache	4KB, 4-way set-associative, 30 cycles latency Same size for Centralized-DSM and Decoupled-DSM.
Permission verification	Register verification latency: 8 cycles
CXL Round Trip Latency	70ns [45]

Workloads. We use eleven workloads from the SPEC 2017 benchmark suite [46], which exhibit diverse memory-access patterns and shared-data parallelism. Each reverse map cache entry handles the permission of a 4KB page. We identify the region of interest (ROI) corresponding to the main OpenMP phase and collect multi-thread instruction traces using Intel Pintool [47]. We fast-forward initialization and simulate 1 billion instructions per core after a 200 million instruction warm-up. We report the LLC Misses Per Kilo Instructions (MPKI) of workloads in Table IV.

Schemes: We compare Decoupled-DSM with five schemes. **Insec-DSM** represents insecure in-place CXL-DSM sharing without TEE protection. **Ideal-DSM** models an upper bound of Decoupled-DSM by assuming perfect permission caching

TABLE IV: Memory intensity of SPEC workloads

Workloads	MKPI	Workloads	MKPI	Workloads	MKPI
603.bwaves_s	41.8	607.cactuBSSN_s	7.1	619.lbm_s	74.4
621.wrf_s	11.9	627.cam4_s	25.1	628.pop2_s	23.6
638.imagick_s	9.4	644.nab_s	6.7	649.fotonik3d_s	55.3
654.roms_s	44.1	657.xz_s	9.9		

and reserved map cache with 100% hit rate. **Centralized-DSM** implements centralized permission enforcement at the CXL side, where all permission checks from different hosts are handled by a CXL centralized authenticator. **Encrypt-Copy-DSM** represents secure sharing via software-based encryption, MAC computation, and data copying through secure channels. **Insec-DE-DSM** implements a Data-Enclave-style [15] design with per-host access control but without software encryption, MAC, or data copying; since it does not support multi-host security, we treat it as insecure.

VIII. Evaluation

A. End-to-end Results

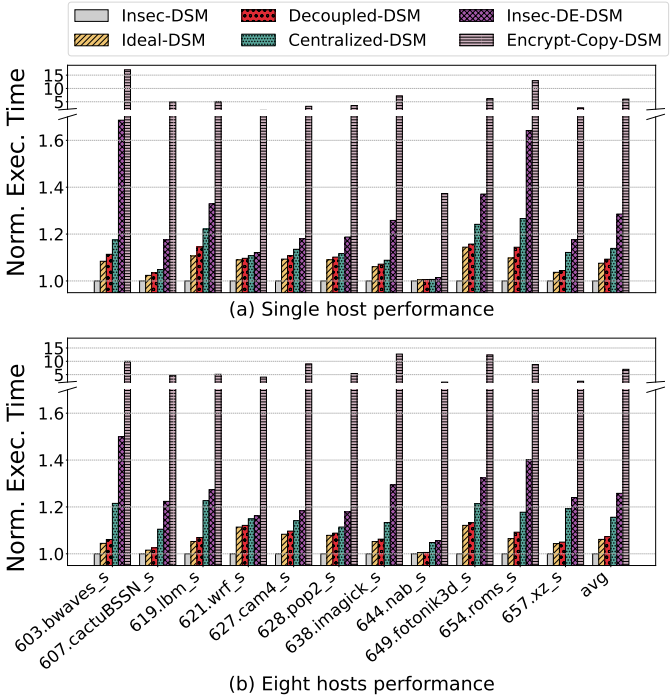


Fig. 11: Normalized execution time under (a) one host and (b) eight hosts, normalized to Insec-DSM.

Figure 11 (a) reports the one-host results. Decoupled-DSM incurs only 9.2% overhead and remains close to Ideal-DSM, as most permission checks hit in the host-side local reverse-map cache. Centralized-DSM incurs 13.9% overhead because every data access requires a CXL-side permission check on the critical path. Insec-DE-DSM incurs 28.5% overhead, while Encrypt-Copy-DSM performs worst, with 6.0× execution time due to encryption, integrity protection, copying, and data re-materialization for cross-host sharing.

Figure 11 (b) shows the eight-host results, where shared-memory bandwidth contention is much higher. Decoupled-DSM still incurs only 7.3% overhead over Insec-DSM and remains close to Ideal-DSM. In contrast, Centralized-DSM degrades due to contention on the centralized permission-checking path and permission-cache bandwidth. Insec-DE-DSM incurs 25.8% overhead, while Encrypt-Copy-DSM is the slowest, with 6.99× overhead on average.

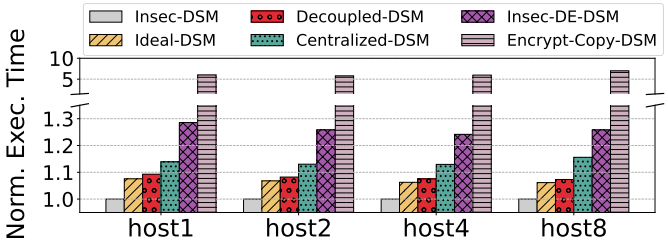


Fig. 12: Avg. normalized execution time across 1–8 hosts.

Figure 12 shows scalability from one to eight hosts. Execution time increases for all schemes due to higher shared-memory bandwidth contention, but their security overheads scale differently. Centralized-DSM is increasingly limited by its centralized permission-check bandwidth, while Encrypt-Copy-DSM suffers from growing shared-data movement. Decoupled-DSM provides the best scalability: across one to eight hosts, it incurs only 7.3%–9.2% overhead over Insec-DSM (8.1% on average) and reduces security overhead by 33.2%–53.1% (41.2% on average) compared with Centralized-DSM.

B. Detailed Analysis and Sensitivity Study

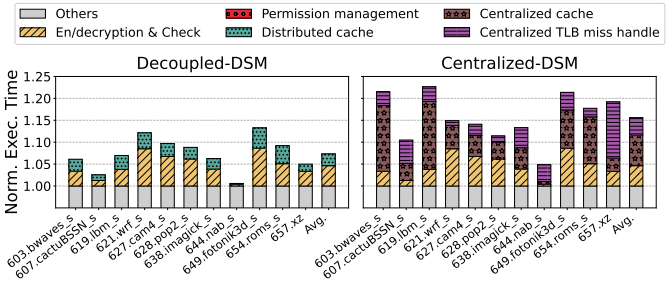


Fig. 13: Overhead breakdown under the 8-host configuration. Results are normalized to Insec-DSM.

We further conduct an overhead breakdown study under the 8-host configuration to break down the overhead of Decoupled-DSM and Centralized-DSM. Figure 13 reports the normalized execution time of incremental designs. With only encryption and permission checks, the average overhead is 4.6% over Insec-DSM, representing the lower-bound security datapath cost. The overhead of management operations is less than 0.1%, showing that these control-path operations are trivial in these workloads because they are much less frequent than data accesses.

Decoupled-DSM incurs 7.3% overhead on average with realistic host-side metadata caches. In contrast, Centralized-DSM increases overhead to 11.6% because permission checks contend for the shared CXL-side cache causing extra CXL round trip and bandwidth limitation. When extra TLB-miss-handling overhead is included, Centralized-DSM reaches 15.6% due to cross CXL link overhead. Compared with this centralized design, Decoupled-DSM reduces security overhead

by 53.1%, demonstrating that moving common-case permission checks to the host-side LCRMT/LCRMC substantially improves scalability.

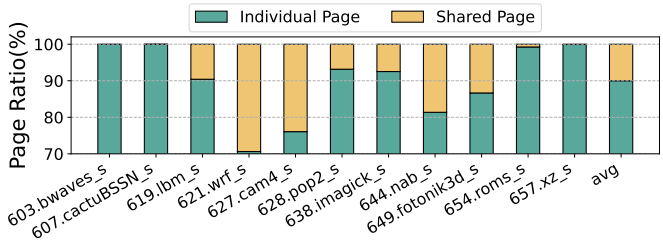


Fig. 14: Ratio of host-private and cross-host shared pages.

1) ***Data-Sharing Characteristics****:* To characterize sharing behavior, we classify pages as *host-private* if accessed by one host and *cross-host shared* if accessed by multiple hosts. Figure 14 shows that, on average, 89.9% of pages are host-private and 10.1% are shared. This favors Decoupled-DSM because most accesses use stable host-local permission checks, while only shared pages may trigger grant/revoke, mmap/munmap, or permission updates.

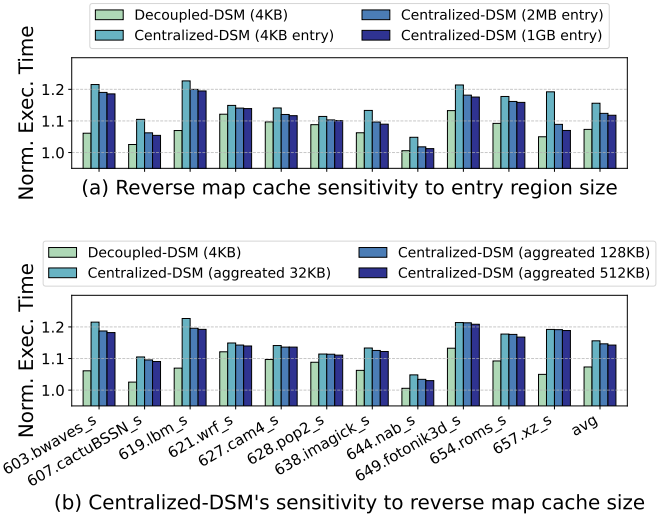


Fig. 15: Decoupled-DSM and Centralized-DSM sensitivity to different cache settings.

2) ***Cache Sensitivity Analysis****:* Figure 15 (a) varies the region size represented by each CXL-side permission-cache entry. Increasing the entry coverage from 4KB to 2MB and 1GB reduces the average normalized execution time of Centralized-DSM from 1.156× to 1.124× and 1.118×, respectively. However, even with 1GB entries, Centralized-DSM still remains slower than Decoupled-DSM, which achieves 1.07× on average. Thus, larger regions reduce metadata fragmentation but do not remove the centralized lookup bottleneck.

Figure 15 (b) further studies Centralized-DSM with larger reverse-map-cache capacities. Increasing the aggregated cache size from 32KB, matching the total 4KB-per-host cache capacity in Decoupled-DSM, to 128KB and 512KB reduces the average normalized execution time of Centralized-DSM from 1.156× to 1.146× and 1.143×, respectively. The improvement is limited because larger capacity only reduces cache misses; it does not remove the shared lookup-throughput bottleneck or the CXL-side validation on the TLB-miss critical path. In contrast, Decoupled-DSM performs common-case permission checks locally through per-host LCRMT/LCRMC, achieving more scalable performance.

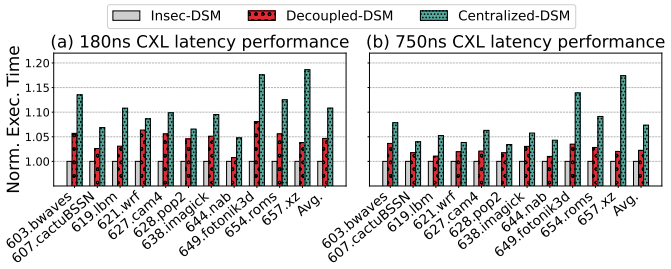


Fig. 16: CXL-latency sensitivity with 8 hosts configuration. Results are normalized to Insec-DSM.

3) CXL-Latency Sensitivity: Figure 16 shows Decoupled-DSM with 180ns [45] and 750ns [48] CXL round-trip latencies, modeling real-world switched-CXL deployments. Higher CXL latency increases the absolute execution time of all schemes, but Decoupled-DSM's normalized overhead over Insec-DSM remains stable and can even slightly decrease. This is because the dominant CXL access latency is shared by both schemes, while most Decoupled-DSM security costs, such as local LCRMC lookups and encryption/decryption, do not scale with CXL latency. Although permission checks and page-fault validation become more expensive, they are infrequent compared with regular memory accesses. Thus, Decoupled-DSM does not introduce a latency-amplifying security path under switched-CXL-like settings.

C. Real-world Application

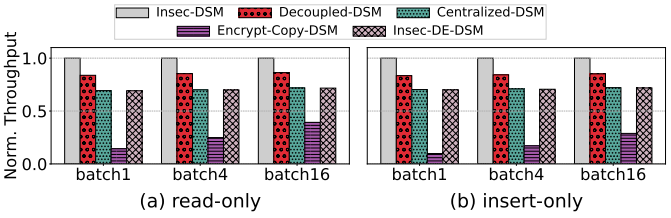


Fig. 17: Redis throughput normalized to Insec-DSM.

We modify Redis [49], a widely used key-value store, to use shared memory for client-server communication. The client writes requests to shared memory every 1/4/16 request(s), and the server writes responses back for the client to retrieve. We evaluate 1M transactions under read-only and insert-only workloads. Figure 17 reports Redis throughput. With security overhead, Decoupled-DSM reduces throughput by 14.9% and 15.8% under read-only and write-only conditions compared to Insec-DSM, and improves throughput by 19.7%, 281.3%, and

20.2% over Centralized-DSM, Encrypt-Copy-DSM, and Insec-DE-DSM, respectively. Although larger batches amortize copy and encryption overheads in Encrypt-Copy-DSM, Decoupled-DSM still achieves significantly higher throughput.

D. Micro-benchmark

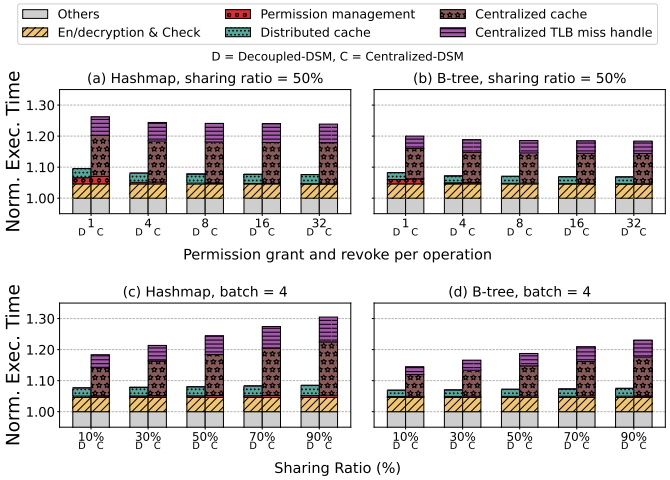


Fig. 18: B-tree and hashmap breakdown of Decoupled-DSM and Centralized-DSM under varying permission-management frequencies and sharing ratios.

Following prior work evaluation about TEE-sharing [15], [25], [50], we use B-tree and hashmap microbenchmarks to evaluate Decoupled-DSM under controlled sharing and management operation frequency. Both workloads run on 8 hosts with a 16GB CXL-resident dataset and 4KB records. We vary the cross-host sharing ratio from 10% to 90% and the sharing-management frequency by triggering a permission grant and revoke per every 1, 4, 8, 16, and 32 operation(s) (search, insert, or update), denoted as batch1 to batch32. Results are normalized to Insec-DSM.

Figure 18 (a)–(b) shows the impact of management-operation frequency under 50% sharing. For hashmap, Decoupled-DSM reduces security overhead from 9.6% at batch1 to 7.6% at batch32, while Centralized-DSM reduces from 26.2% to 23.9%. For B-tree, Decoupled-DSM reduces from 8.3% to 6.9%, while Centralized-DSM reduces from 20.0% to 18.4%. This shows that batching amortizes permission-management overhead, but Decoupled-DSM remains consistently closer to Insec-DSM. Figure 18(c)–(d) shows the impact of sharing ratio under batch4. For hashmap, Decoupled-DSM increases only from 7.6% to 8.5% as the sharing ratio grows from 10% to 90%, while Centralized-DSM increases from 18.3% to 30.5%. For B-tree, Decoupled-DSM increases from 6.9% to 7.5%, while Centralized-DSM increases from 14.5% to 23.1%. Overall, Decoupled-DSM incurs 7.7% overhead on average, compared with 21.7% for Centralized-DSM. In summary, the results show that Decoupled-DSM gains good scalability when the sharing-management frequency increases, and it is less sensitive to the sharing ratio than sharing-management frequency because steady-state permission checks are served locally through LCRMT/LCRMC.

IX. Related Work

Confidential Computing and TEEs: Modern confidential cloud computing is heavily anchored by CPU-centric TEEs, such as Intel SGX [23], Intel TDX [5], and AMD SEV-SNP [4]. A majority of literature has focused on minimizing the operational overheads of these environments and enhancing intra-host memory protection [51]–[56]. However, these architectures are fundamentally predicated on the assumption that memory is tightly coupled to and exclusively controlled by a single CPU package [15], [25], [27], [57]. While these efforts substantially advance performance and usability, they typically remain within a single-host trust boundary where the CPU package is the only authority that can validate memory ownership and enforce access control.

CXL-DSM and Memory Pooling: Recent systems research has explored CXL for memory disaggregation, pooling, and tiering to improve resource utilization [45], [58], [59]. Advanced distributed systems such as Tigon exploit CXL-DSM's coherent, in-place sharing to accelerate distributed databases [10], [12], [60], [61]. However, these systems typically assume trusted hosts or rely on coarse-grained fabric partitioning, and thus do not provide TEE-grade isolation against compromised hypervisors [62], [63].

X. Conclusion

This paper presents Decoupled-DSM TEE, a scalable architecture for secure multi-host CXL disaggregated shared memory. By decoupling memory allocation from access enforcement and introducing a host-CXL validation protocol, Decoupled-DSM bridges the gap between CPU-centric TEEs and CXL-DSM while preventing unauthorized cross-host access. Evaluation results show that it provides strong security with low overhead, achieving near-baseline performance and outperforming centralized and encrypt-copy designs.

Acknowledgment

We sincerely thank the anonymous reviewers from MICRO 2026 for their insightful feedback. This work is supported in part by the National Science Foundation (NSF) under Grant No. IIS-2543427. Any opinions, findings, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] Amazon, "Aws nitro enclaves," https://aws.amazon.com/ec2/nitro/nitro-enclaves, 2024.

[2] Microsoft, "Azure confidential computing," https://learn.microsoft.com/en-us/azure/confidential-computing/, 2024.

[3] Google, "Google confidential computing," https://cloud.google.com/security/products/confidential-computingdocumentation, 2024.

[4] AMD, "Amd sev-snp," https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf, 2022, online; accessed Mar, 2026.

[5] Intel, "Intel trust domain extensions (intel tdx)," *https://www.intel.com/content/developer/articles/technical/intel-trust-domain-extensions.html*, 2023.

[6] A. Limited, "Arm confidential compute architecture," Arm Limited, 2023. [Online]. Available: https://www.arm.com/architecture/security-features/arm-confidential-compute-architecture

[7] S.-Y. Tsai, Y. Shan, and Y. Zhang, "Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated {Key-Value} stores," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 33–48.

[8] T. Ma, Z. Liu, C. Wei, J. Huang, Y. Zhuo, H. Li, N. Zhang, Y. Guan, D. Niu, and M. Zhang, "Hydrarpc: Rpc in the cxl era," in *2024 USENIX Annual Technical Conference (USENIX ATC 24)*, 2024, pp. 387–395.

[9] C. Alverti, S. Psomadakis, B. Ocalan, S. Jaiswal, T. Xu, and J. Torrellas, "Cxlfork: Fast remote fork over cxl fabrics," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2025, pp. 210–226.

[10] Y. Huang, H. Chen, N. Ni, V. Chidambaram, D. Tang, E. Witchel, Z. Zhu, and Z. Jia, "Tigon: A distributed database for a cxl pod," in *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25), Boston, MA*, 2025.

[11] D. S. Berger, Y. Zhong, F. Kazhamiaka, P. Zardoshti, S. Teng, M. D. Hill, and R. Fonseca, "Octopus: Scalable low-cost cxl memory pooling," *arXiv preprint arXiv:2501.09020*, 2025.

[12] X. Yang, Y. Zhang, H. Chen, F. Li, G. Fan, Y. Kong, B. Wang, J. Fang, Y. Wang, and T. Huang, "Unlocking the potential of cxl for disaggregated memory in cloud-native databases," in *Companion of the 2025 International Conference on Management of Data*, 2025, pp. 689–702.

[13] Intel Corporation, "3rd gen intel xeon scalable processors brief," https://www.intel.com/content/products/docs/processors/xeon/3rd-gen-xeon-scalable-processors-brief.html, Apr 2021, online; accessed Mar, 2026.

[14] AMD, "Amd sev," https://developer.amd.com/sev/, 2022, online; accessed Mar, 2026.

[15] Y. Xu, J. Pangia, C. Ye, Y. Solihin, and X. Shen, "Data enclave: A data-centric trusted execution environment," in *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2024, pp. 218–232.

[16] T. P. Morgan, "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//, 2021, online; accessed Mar, 2026.

[17] C. Mellor, "Intel sees cxl as rack-level disaggregator with optane connectivity," https://blocksandfiles.com/2021/08/18/intel-sees-cxl-as-rack-level-disaggregator/, Aug 2021, online; accessed Mar, 2026.

[18] Samsung, "Samsung cxl solutions – cmm-h," https://semiconductor.samsung.com/us/news-events/tech-blog/samsung-cxl-solutions-cmm-h/, 2021, online; accessed Mar, 2026.

[19] ——, "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, 2021, online; accessed Mar, 2026.

[20] Y. Sun, Y. Yuan, Z. Yu, R. Kuper, C. Song, J. Huang, H. Ji, S. Agarwal, J. Lou, and I. Jeong, "Demystifying cxl memory with genuine cxl-ready systems and devices," in *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 2023, pp. 105–121.

[21] CXL, "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/, 2024.

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

[23] Intel Corporation, "Intel software guard extensions (Intel SGX)," https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html, 2022, online; accessed Mar, 2026.

[24] A. Kalia, M. Kaminsky, and D. G. Andersen, "Design guidelines for high performance rdma systems," in *2016 USENIX Annual Technical Conference (USENIX ATC 16)*, 2016, pp. 437–450.

[25] J. Z. Yu, S. Shinde, T. E. Carlson, and P. Saxena, "Elasticlave: An efficient memory model for enclaves," in *31st USENIX Security Symposium (USENIX Security 22)*. USENIX, 2022, pp. 4111–4128.

[26] N. Gober, G. Chacon, L. Wang, P. V. Gratz, D. A. Jimenez, E. Teran, S. Pugsley, and J. Kim, "The championship simulator: Architectural simulation for education and competition," *arXiv preprint arXiv:2210.14324*, 2022.

[27] J. Shi, J. Gu, Y. Xia, and H. Chen, "Serverless functions made confidential and efficient with split containers," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 1091–1110.

[28] Compute Express Link Consortium, "Compute express link (cxl) specification, revision 3.0," Compute Express Link Consortium, Specification, August 2022. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[29] ——, "Compute express link (cxl) specification, revision 3.1," Compute Express Link Consortium, Specification, November 2023. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[30] ——, "Compute express link (cxl) specification, revision 4.0," Compute Express Link Consortium, Specification, November 2025. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[31] J. Wang, A. Li, H. Li, C. Lu, and N. Zhang, "Rt-tee: Real-time system availability for cyber-physical systems using arm trustzone," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 352–369.

[32] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Commun. ACM*, vol. 52, no. 5, p. 91–98, May 2009. [Online]. Available: https://doi.org/10.1145/1506409.1506429

[33] D. Micciancio and M. Walter, "On the bit security of cryptographic primitives," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 3–28.

[34] F.-X. Standaert, "Introduction to side-channel attacks," in *Secure integrated circuits and systems*. Springer, 2009, pp. 27–42.

[35] D. Lewine, *POSIX programmers guide*. " O'Reilly Media, Inc.", 1991.

[36] B. Nichols, D. Buttlar, and J. Farrell, *Pthreads programming: A POSIX standard for better multiprocessing*. " O'Reilly Media, Inc.", 1996.

[37] D. R. Butenhof, *Programming with POSIX threads*. Addison-Wesley Professional, 1993.

[38] M. K. McKusick, W. N. Joy, S. J. Leffler, and R. S. Fabry, "A fast file system for unix," *ACM Transactions on Computer Systems (TOCS)*, vol. 2, no. 3, pp. 181–197, 1984.

[39] N. Provos, M. Friedl, and P. Honeyman, "Preventing privilege escalation," in *12th USENIX Security Symposium (USENIX Security 03)*, 2003.

[40] J. Gu, B. Zhu, M. Li, W. Li, Y. Xia, and H. Chen, "A {Hardware-Software} co-design for efficient {Intra-Enclave} isolation," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 3129–3145.

[41] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "Cacti 5.1," Technical Report HPL-2008-20, HP Labs, Tech. Rep., 2008.

[42] Compute Express Link Consortium, "Compute express link (cxl) specification, revision 2.0," Compute Express Link Consortium, Specification, November 2020. [Online]. Available: https://computeexpresslink.org/cxl-specification/

[43] Rambus, "CXL 2.0 Controller," https://www.rambus.com/interface-ip/cxl/cxl2-controller/, 2026, accessed: 2026-06-16.

[44] S. Park, S. Lee, W. Xu, H. Moon, and T. Kim, "libmpk: Software abstraction for intel memory protection keys," in *2019 USENIX Annual Technical Conference (USENIX ATC 19)*, 2019, pp. 241–254.

[45] H. Li, D. S. Berger, L. Hsu, D. Ernst, P. Zardoshti, S. Novakovic, M. Shah, S. Rajadnya, S. Lee, and I. Agarwal, "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*, 2023, pp. 574–587.

[46] Intel, "Spec cpu 2017." https://www.spec.org/cpu2017/, 2017, online; accessed Mar, 2026.
[47] M. Bach, M. Charney, R. Cohn, E. Demikhovsky, T. Devor, K. Hazelwood, A. Jaleel, C.-K. Luk, G. Lyons, and H. Patil, "Analyzing parallel programs with pin," *Computer*, vol. 43, no. 3, pp. 34–41, 2010.
[48] X. Yang, Q. Hu, J. Li, F. Li, Y. Zhu, Y. Zhou, Q. Lin, J. Dai, Y. Kong, J. Zhang *et al.*, "Beluga: A cxl-based memory architecture for scalable and efficient llm kvcache management," *Proceedings of the ACM on Management of Data*, vol. 4, no. 1 (SIGMOD, pp. 1–29, 2026.
[49] S. Sanfilippo, "Redis," https://redis.io/, 2026, online; accessed Mar, 2026.
[50] J. Park, N. Kang, T. Kim, Y. Kwon, and J. Huh, "Nested enclave: Supporting fine-grained hierarchical isolation with sgx," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 776–789.
[51] C.-C. Tsai, D. E. Porter, and M. Vij, "Graphene-sgx: A practical library os for unmodified applications on sgx," in *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, 2017, pp. 645–658.
[52] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, and M. L. Stillwell, "Scone: Secure linux containers with intel sgx," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 689–703.
[53] Y. Shen, H. Tian, Y. Chen, K. Chen, R. Wang, Y. Xu, Y. Xia, and S. Yan, "Occlum: Secure and efficient multitasking inside a single enclave of intel sgx," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 955–970.
[54] S. Shinde, D. Le Tien, S. Tople, and P. Saxena, "Panoply: Low-tcb linux applications with sgx enclaves." in *NDSS*, 2017.
[55] O. Weisse, V. Bertacco, and T. Austin, "Regaining lost cycles with hotcalls: A fast interface for sgx secure enclaves," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 2, pp. 81–93, 2017.
[56] M. Taassori, A. Shafiee, and R. Balasubramonian, "Vault: Reducing paging overheads in sgx with efficient integrity verification structures," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, 2018, pp. 665–678.
[57] S. Zhao, P. Xu, G. Chen, M. Zhang, Y. Zhang, and Z. Lin, "Reusable enclaves for confidential serverless computing," in *32nd USENIX security symposium (USENIX Security 23)*, 2023, pp. 4015–4032.
[58] D. S. Berger, D. Ernst, H. Li, P. Zardoshti, M. Shah, S. Rajadnya, S. Lee, L. Hsu, I. Agarwal, and M. D. Hill, "Design tradeoffs in cxl-based memory pools for public cloud platforms," *IEEE Micro*, vol. 43, no. 2, pp. 30–38, 2023.
[59] J. Jang, H. Choi, H. Bae, S. Lee, M. Kwon, and M. Jung, "Cxl-anns: Software-hardware collaborative memory disaggregation and computation for billion-scale approximate nearest neighbor search," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 585–600.
[60] Y. Guo and G. Li, "A cxl-powered database system: Opportunities and challenges," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE, 2024, pp. 5593–5604.
[61] M. Ahn, A. Chang, D. Lee, J. Gim, J. Kim, J. Jung, O. Rebholz, V. Pham, K. Malladi, and Y. S. Ki, "Enabling cxl memory expansion for in-memory database management systems," in *Proceedings of the 18th International Workshop on Data Management on New Hardware*, 2022, pp. 1–5.
[62] K. Choi, I. Kim, S. Lee, and J. Huh, "Shieldcxl: a practical obliviousness support with sealed cxl memory," *ACM Transactions on Architecture and Code Optimization*, vol. 22, no. 1, pp. 1–25, 2025.
[63] X. Wang, J. Kotra, A. Jones, W. Xiong, and X. Jian, "Counter-light memory encryption," in *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2024, pp. 724–738.