Confidential Computing with Heterogeneous Devices at Cloud-Scale

Aritra Dhar[†] Supraja Sridhara[‡] Shweta Shinde[‡] Srdjan Capkun[‡] Renzo Andri[†]

[†]Computing Systems Lab, Huawei Zurich Research Center [‡]ETH Zürich

***Abstract*—Cloud-centric workloads increasingly leverage domain-specific accelerators (DSAs) such as GPU, NPU, FPGA, etc., to achieve massive speedup over general-purpose CPUs. These workloads compute sensitive data; furthermore, the programs can be proprietary business secrets such as high-performance AI models. Therefore, several confidential cloud solutions have recently emerged to protect against the attacker-controlled software stack (OS/VMM) and the cloud service providers or CSPs themselves. CPU-centric trusted execution environments, or TEEs, have been around for decades and are deployed commercially. However, despite some recent proposals, most nodes lack TEE capability and, therefore, are unprotected against malicious CSP and software stack.**

We address this gap by proposing a new dedicated hardware module, the security controller (SC), that acts as the TEE proxy for the legacy non-TEE DSA nodes in a data center across racks. SC enforces access control and attestation mechanisms and protects the non-TEE nodes even from a physical attacker. This way, SC enables new-generation TEE-enabled nodes and legacy non-TEE nodes to be used in a data center simultaneously while ensuring security. We implement and synthesize SC hardware and evaluate it with real-world cloud-centric workloads with heterogeneous DSAs. Our evaluation shows that, on average, SC introduces 1.5-5% overhead while running AI, Redis, and file system workloads and scales well with an increasing number of DSA nodes (up to 2236 concurrent NPUs running CNNs).

I. Introduction

Cloud service providers (CSP) are increasingly using heterogeneous *domain-specific accelerators* or DSAs [1, 2, 3] to accelerate workloads such as AI/ML [4], graph processing [5], in-memory caches [6], and many more. To maximize resource utilization and scalability, DSAs are shared between multiple tenants concurrently [7]. Such heterogeneity brings additional challenges as the attack surface includes mutually distrusting co-tenants, as well as CSPs. A malicious CSP controls all the software and hardware infrastructure and can maliciously modify the configuration of tenants' nodes [8]. Trusted execution environments (TEEs) for CPUs [9, 10] as well as for DSAs [11, 12, 13, 14] enable secure user code execution without trusting the OS or hypervisors. CPU-TEEs have become important offerings in the cloud [15, 16, 17, 18]. Although TEEs serve a necessary role, they are not sufficient if the user wants to fully use the heterogeneity in a modern multi-tenant cloud.

Only a small fraction of the data center nodes that tenants can rent are TEE-enabled. If users want to accelerate training/inference on an AI accelerator, they have to choose between performance and security. Either the users execute the computation on a TEE-enabled general-purpose CPU, therefore sacrificing performance. Or, the users execute on the non-TEE AI accelerator in an unprotected environment, risking compromising the data and computation. One way to address this problem is to make all nodes TEE-enabled. However, this is impractical and leaves out many non-TEE legacy devices. Prior works protect non-TEE nodes by deploying bus-level isolation if they are all physically connected to a TEE host [19, 20, 21]. Such host-centric solutions are not applicable to a disaggregated data center setting, where similar types of nodes are clustered together in racks. The racks are connected over high-speed switches, as shown in Fig. 1. A CSP may have several such data centers in different geographical locations connected over fast networking. A user workload may span over multiple data centers. Solutions that enable TEE abstractions for a single rack containing non-TEE CPU-GPU seem promising [22]. However, they do not scale to multiple racks, do not support multi-tenancy, require changes in the software stack, and are not designed to leverage TEE-enabled nodes. Lastly, simply connecting TEE nodes and rack-level solutions is not sufficient, as it does not make them collectively secure. Thus, careful design considerations are required for security and scalability at a data center level.

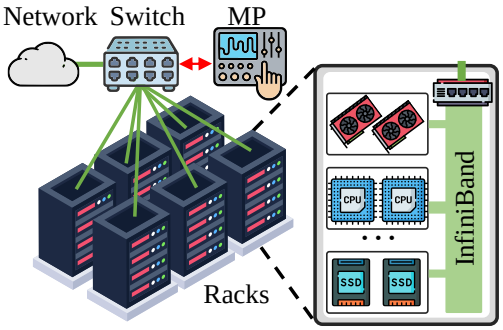

Fig. 1: The figure shows a high-level overview of Data center topology. Racks are connected over high-speed optical switches, while clusters of nodes within a rack are linked via InfiniBand to the data center-level switch. The management plane (MP) performs resource management and monitoring.

To address this gap, in this paper, we design a solution to protect legacy non-TEE DSA nodes and connect them to the new-generation TEE-capable nodes. We design a distributed TEE solution that allows a tenant to securely use TEE nodes

(both CPUs and DSAs) and non-TEE legacy nodes. We use three main insights to achieve this goal. *First*, we use TEEs on available CPUs and DSAs to protect all outgoing data. *Second*, we employ a security controller (SC), a dedicated low-TCB hardware device, to protect all non-TEE nodes. The SC provides TEE properties such as attestation, isolation, and secure channels to all non-TEE nodes. Although these two design decisions may appear trivial and adequate to protect data and execution on nodes, further examinations reveal several issues that require attention. For example, the CSP can reconfigure nodes such that multiple tenants are mapped to the same memory location, compromising isolation or disabling TEE protection during sensitive computations. A physical attacker can bypass the security controller by, e.g., directly connecting non-TEE nodes physically. Local protection at each node is insufficient, and we must ensure global isolation. Our *third* insight systematically addresses the gaps by attesting and locking the initial state of nodes and checking resource management decisions for violations of resource isolation and secure path guarantees. We provide a secure physical perimeter to non-TEE nodes for protection against physical attacks. In summary, we approach the problem in a distributed manner by first ensuring local security and then enforcing global properties to maintain security. We synthesize and place-and-route (where the components/blocks are placed and wires are routed between them) our prototype SC hardware in 28 nm, which is 0.46 mm^2. Our synthesized SC prototype scales with concurrent multi-tenants and supports up to 2236 concurrent NPUs running RetinaNet-RN50 or up to 117 SSDs running the FIO (R-RW) benchmark. We use three cloud-focused benchmarks (CNN inference, Redis, and file system) to show the practicality of our setup with low overhead.

Our Contributions. We now summarize our contributions:

1) **Design Space Exploration** We explore the design space for enclave execution in a heterogeneous setting of a multi-tenant cloud and explore five potential ways to compose distributed enclaves with a mixture of TEE and non-TEE nodes. We also analyze the pros and cons of these design choices and pick the most suitable for large-scale data centers. We show the feasibility of using TEE and non-TEE nodes to provide enclave security primitives (Section III).
2) **End-to-end System Design and Analysis** Our end-to-end system for data centers include both TEE and non-TEE nodes. We enable TEE in two new DSAs, use two existing TEE-DSAs, and integrate them with our system. Our attestation mechanisms protect the integrity of distributed enclaves, as well as the configurations of the platforms involved. We provide an extensive security analysis of our proposed design against potential attacks from untrusted cloud service providers and compromised software stacks (Section IV, and V).
3) **Hardware Prototype and Evaluation** We implement our proposed design in dedicated hardware and evaluate real-world cloud-centric workloads such as AI, storage, and in-memory databases that show our system's scalability and modest performance costs (Section VI).

II. Problem Statement

We motivate the need for a new cloud-scale design that can support confidential computing for a diverse set of devices, with and without TEE support.

A. Setting

Disaggregated architecture [23] in modern data centers leverages the performance of domain-specific accelerators (DSA) such as GPUs, NPUs, storage [24], and smart NICs [25] connected over a fast interconnect as shown in Fig. 1. Each rack has several CPU or DSA nodes where the CPU nodes are TEE-capable, and a DSA node could either be a TEE or a non-TEE (legacy) node. The cloud service provider (CSP) uses a management plane (CSP-MP) for resource allocation, revocation, and monitoring [26, 27, 28]. In this setting, a tenant deploys a job by submitting a manifest to the CSP-MP, specifying the required size and type of resources from individual nodes.

Motivation: TEE and Non-TEE Devices in Data Centers. A typical data center workload could be training a large language model with a large data set or running an inference job (such as a chatbot) that receives a large set of user queries. This would involve CPU nodes running the software stack (OS, TensorFlow, PyTorch), multiple GPUs and FPGAs accelerating the training and inference, and SSDs storing the training/inference data and results. We assume that in terms of TEE-capability, the nodes are heterogeneous. For example, we can assume that most of the CPUs are TEE enabled as most of the major hardware vendors such as Intel [29], AMD [10], and ARM [30] provide mature VM-based TEEs. The rest of the DSAs can either be TEE-capable or legacy devices. While some existing GPUs, such as Nvidia H100 [12], provide confidential computing features, most other GPUs, NPUs, SSDs, FPGA, and many more are not TEE-capable. While the TEE-capable devices can isolate multiple tenants and provide a secure channel, the non-TEE nodes do not have such a feature.

B. Trusted Execution Environments (TEEs)

TEEs provide sandboxed execution domains against an attacker-controlled software stack such as OS and hypervisor. TEEs could be standalone programs [31] or fully-fledged operating systems [30, 10, 29]. Due to the large diversity of TEE designs and their security properties, formally defining the TEEs is tricky. For example, in Intel SGX, any IO mechanisms are not secure as SGX enclaves cannot make system calls and are solely handled by the privileged OS. Therefore, secure IO is not SGX's security property. However, VM-based TEEs such as AMD SEV [10] and Intel TDX [29] have extensive security features to protect IO interfaces [32]. Existing work categorizes hardware-supported TEEs based on their security features, such as memory and cache isolation, attestation, secure IO, etc [33]. To define TEEs, we rely on existing TEE definitions in the literature [34], where TEE enclaves are abstracted to *Separation Kernels*, which is defined in the Separation Kernel Protection Profile (SKPP) [35]. A separation kernel is an isolated execution with carefully controlled interfaces to communicate with the external world or other separation kernels. A separation kernel may have shared resources

with other kernels. The security properties of separation kernels can be found in literature [34] and can be listed as the following:

- *Spatial isolation:* Data between the separation kernels are completely isolated (read/write/execute).
- *Temporal isolation:* The shared resources between separation kernels cannot leak information to each other.
- *Control of information flow:* All the communication interfaces between the separation kernels are vetted and cannot be performed unless explicitly permitted.
- *Fault isolation:* A fault or a security vulnerability in one separation kernel stays localized and cannot be spread to others.

The definition of a TEE relies on the assumption of trust, typically both static and dynamic trust. Static trust is a comprehensive security evaluation based on security requirements carried out only once before deployment. Meanwhile, dynamic trust is based on an immutable trust anchor known as the root of trust (RoT). RoT can be a secure hardware key in a tamper-proof chip. RoT is responsible for measuring the changing system states (e.g., loading a kernel module) and is carried out through the system lifecycle.

In summary, a TEE ensures the following:

1) *Security in execution:* The integrity and confidentiality of the data and program remain protected from the attacker. There could be a bootstrapping mechanism, such as a measured boot, to ensure that the TEE has been initialized properly or the correct CPU firmware is present.
2) *Security in transit:* Communication between enclaves or between enclave and untrusted environment is mediated. This also includes secure IO paths, such as communication with peripheral devices.
3) *Security in storage*: Secure storage is necessary to ensure that the integrity and confidentiality of the data or keys are protected while at rest. This also includes rollback prevention.

C. Threat Model

Enclave codes running on CPU cores and DSAs are trusted to run a job. Co-located enclaves from different tenants are mutually distrusting. The CSP deploys all the nodes in the data center. We trust the hardware manufacturer for TEE nodes. These nodes have a hardware root of trust for generating correct attestation reports and securing keys. We assume an up-to-date revocation list is available to determine if an attestation report is from a revoked node. After verifying its integrity, we also trust the node's TEEs software TCB (e.g., VMPL0 in AMD SEV), including the firmware. We assume that host OS, hypervisors, CSP-MP, and co-located tenants are malicious. We assume a physical attacker that controls all nodes (including the BIOS and chipset), can plug in and out malicious nodes and network devices (e.g., switches, routers, network interconnects, etc.), and can probe the bus and manipulate all network traffic, therefore, rendering PCIe link encryption and device certification mechanisms [36] untrustworthy. Finally, we leave Iago [37], side-channels, hardware trojans, and denial-of-service out of scope for this paper.

D. Gap in Prior Works

To the best of our knowledge, HETEE [22] is the only proposal that addresses rack-scale confidential computing infrastructure. Despite addressing similar problem space, there are key fundamental points that set us apart from HETTE. In HETEE, the proxy node OS is reloaded before starting the next job to ensure the node is secure. Furthermore, the boot ROM chip on the proxy node is removed and replaced with the PCB traces connected to the SC. This enables the SC to monitor the secure boot process in the node. The secure boot only ensures the proxy node can be loaded with the signed OS image. However, it does not ensure that the OS has no existing vulnerabilities. Therefore, to ensure the job is isolated on the proxy, one must trust that the proxy node OS will behave correctly, i.e., the OS in the TCB. Contrary to this, our design excludes the OS and hypervisor from the TCB as we do not allow any CPU (and, therefore, any OS) as a non-TEE node connected to our SC. A proxy node and corresponding GPU are explicitly allocated for a single tenant. Therefore, no multi-tenancy is possible. As the accelerator is physically attached to the proxy node (i.e., the node with the CPU that is the primary controller of the accelerator) behind the SC, there is no way to assign multiple accelerators to that proxy node dynamically. For example, a host cannot be assigned to a different accelerator without completely switching to its corresponding proxy node. In HETEE, the SC is always on the data path, whereas our design only uses SC on the data path between the TEE and non-TEE nodes. This has a performance implication for HETEE (e.g., GoogleNet in HETEE has over 2× overhead). Moreover, HETEE modifies the software stack. A special program has to perform code and data handover between the server node and proxy node via the SC. This makes HETEE impractical for real-world deployment. We demonstrate unmodified workloads running on our proposed system with low overhead (Sec. VI). Additionally, we discuss other drawbacks of HETEE in Section Sec. III-A, where we explore several potential design choices to contrast distributed enclaves in data centers involving TEE and non-TEE nodes. Other proposals [19, 20, 21] extend CPU-TEE protection by leveraging MMU to protect MMIO devices and perform bus-level isolation since devices are physically connected to a TEE host. However, these do not address the challenges of setting up data center-scale infrastructure with TEE and legacy devices.

III. Distributed TEE Design Space Exploration

We first look at potential design options and draw insights from our observations of the design space exploration.

A. Potential Designs

We explore several potential ways to design a cloud-scale trusted computing infrastructure. A high-level overview of five such designs (S_{0-4}) can be seen in Fig. 2. Security controller (SC) is a part of some of these designs that enforce access control on data between nodes and remote users. SC can be a dedicated hardware module that can be implemented in many ways. We discuss one such design in Sec. IV. In the following section, we explore the constructions of these designs, their security properties, and their advantages and disadvantages.

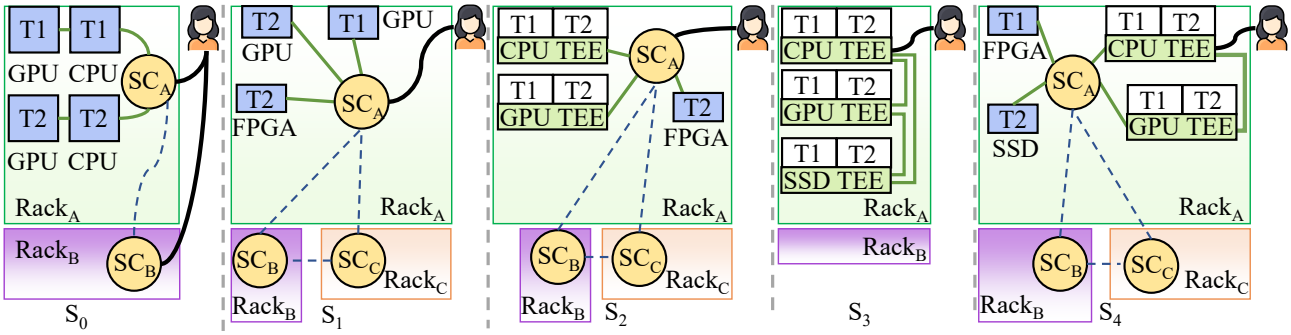

Fig. 2: **Cloud-scale heterogeneous TEE designs**. The figure shows five potential design choices: S_0,S_1,S_2,S_3, and S_4 to construct a distributed TEE with TEE and non-TEE nodes across multiple racks (three in this figure where SC_A,SC_B,and SC_C are located) in a data center. Every rack has its corresponding security controller (SC) that manages the compute nodes in that rack. Cross-SC communication channels facilitate resource allocation across racks for higher scalability. In our design (S_4), we assume the racks may not be physically co-located in the same data center. Our design is agnostic of the communication channel (e.g., Ethernet, InfiniBand) between the SCs across multiple data centers.

1) S_0: Centralized SC connected to nodes managing devices: In this scenario, the user themselves or via some untrusted server can connect to a rack-level SC. The SC transfers the encrypted workload to a CPU node that manages devices. In this case, the tenant fully occupies nodes behind the SC. The SC can only connect to a limited number of CPU nodes, as it intercepts and executes cryptographic operations on all packets to and from the user. HETEE [22] follows such a design principle where all the nodes behind SC are connected physically and located in a single container with PCIe switches. The SC executes remote attestation of the trusted CPU nodes known as proxy nodes. In secure mode, the SC transfers the encrypted workloads (GPU kernels and data) from the untrusted server to the proxy nodes. Proxy nodes then deploy the kernels to their connected GPUs. After computation, proxy nodes send the results to the SC. The SC encrypts the results and sends them to the untrusted server. In insure mode, the SC allows the untrusted servers to connect directly to the GPUs for performance improvements.

This design *does not* provide multi-tenancy or virtualization of devices as the nodes are assigned exclusively to a single tenant. Scaling is challenging as the SC only connects to a limited number of nodes inside a rack. It is also not possible to connect to other racks or nodes of other SCs. HETEE [22] proposes the nodes (host or proxy node and associated GPUs) behind the SC to be enclosed inside a physically hardened case that can prevent physical attackers. Provisioning the GPUs outside the proxy node is possible. However, such a mechanism requires all the security mechanisms to be revoked. Such design decisions and attacker models are impractical, and their limitations make it infeasible for multi-tenant data centers.

2) S_1: Centralized SC connected to nodes with no TEE: Here, the SC directly connects and manages all the nodes. This mitigates some of the disadvantages of S_0: a host CPU can connect to any device, and we can support node-level virtualization. However, as the nodes do not support TEE, the devices must be owned exclusively by *one* tenant, as there is no memory isolation inside a node. All the setup and access control is enforced by the SC, which enforces bus-level filtering so that a tenant cannot access the devices of other tenants.

However, in this design, the SC intercepts all the traffic from the user and between the nodes. Making them a single point of failure and hard to scale. The nodes still need to support some form of identification and secure boot to ensure their integrity. Such a design also does not allow the multi-tenancy we want to achieve.

3) S_2: Semi-centralized SC with TEEs: Here, we consider a setting where some devices can support TEE while the rest do not. TEE nodes support memory isolation, attestation, and sealing on the TEE nodes. The SC is still in charge of managing the devices and communicating with the user. The attestation mechanism to verify the integrity of nodes and enclaves can be offloaded to the trusted SC.

This design solves many of the drawbacks of both S_1 and S_2. Specifically, it allows tenants to be securely isolated on devices that support TEE. SC uses bus filtering for the TEE-less nodes. However, SC still handles the data between the user and the host enclaves. End-to-end authenticated encrypted channel between the nodes' software eliminates the need to trust SC.

4) S_3: Decentralized TEEs: This design eliminates the need for an SC by requiring that all nodes support some hardware TEE features, such as secure boot, attestation, and memory isolation. Such nodes enable fully decentralized TEE where the confidential VMs on the nodes set up the channels between the nodes (by setting up MMIO and DMA channels). The user can directly communicate with the node without needing the SC, eliminating potential bottlenecks. Such a decentralized design also allows the TEE nodes to be virtualized and assigned to any host node. TEE on individual nodes will allow the CSP and all the software stacks to be attacker-controlled.

This TEE design has the most security and scalability advantages compared to the previous three designs and is suitable for modern multi-tenant data centers. However, such a design has a strong requirement: hardware TEE features in all the nodes, which may seem infeasible for many devices, including legacy DSAs.

5) S_4: Hybrid Decentralized TEEs: Given the strong requirements of S_3, we can now design a hybrid model of S_2 and S_3. The hybrid design accommodates new and upcoming TEE-capable DSAs as well as legacy DSAs and other devices without hardware TEE capability. The TEE-capable devices

can be completely decentralized as S_3. Therefore, the enclaves deployed there can leverage virtualization and scale across many racks. The SC allocates the isolated multi-tenant units on the nodes to different tenants who can communicate with these units without going through the SC. Moreover, we can also allocate TEE-less devices. As these devices do not have any isolation or attestation capability, the enclave on the TEE node communicates through the SC, which provides bus filtering for the TEE-less nodes. These devices can be connected physically with the SC. SC acts as the security monitor, communicating the TEE needs over a secure channel and filtering the traffic to the non-TEE nodes based on access policies.

B. Observations

Out of the five design decisions we discussed in the previous section, we chose the last design, S_4, as it provides the most security and functionality suitable for a multi-tenant cloud. Next, we point out our observations alongside the requirements for the nodes necessary for such a design choice.

→ *Observation 1*: Modern DSAs are highly configurable, i.e., the CSP or hypervisor can push configuration to the DSAs' firmware. E.g., the MIG partition on NVIDIA A100 [38] allows reconfiguration (partition size, GPU pass-through, containers), reconfigurable cache [8] can change caches from private to shared, etc. Therefore, monitoring data on the bus is insufficient to determine the local node's configuration state.

→ *Requirement 1*: A secure/measured boot ensures the firmware and device configuration are correct. As attacker-controlled OS/hypervisors can reconfigure a device or flash its firmware, it is crucial to ensure that the device is initialized from a safe state. This observation applies to both TEE and non-TEE-capable devices, as either the user or the SC needs to ensure that the integrity of the device firmware is maintained before secret provisioning and does not change later.

→ *Observation 2*: Centralized SC is unaware of the DSAs' execution due to their asynchronous nature. Therefore, any misbehaving program on a DSA can manipulate or access the data of other tenants if the DSA does not implement its own memory isolation mechanism.

→ *Requirement 2*: Attestation ensures that the DSAs are running the proper code (also known as the device kernels). *Runtime isolation* ensures that the attacker cannot access the memory of other processes. *Trusted path* ensures the communication channel between the host and the device is protected, i.e., the attacker cannot manipulate the data on the bus. *Sealing* ensures that the integrity and confidentiality of the data on persistent storage (i.e., an SSD) is protected from the attacker.

→ *Observation 3*: Non-TEE nodes do not support isolated multi-tenancy without a hypervisor. Therefore, one user must have exclusive access to a non-TEE node behind SC. Moreover, the non-TEE nodes do not support encryption to protect the data on the fly.

→ *Requirement 3*: SC must ensure a unique binding from a TEE node to a set of non-TEE nodes connected to the SC. After the non-TEE nodes are allocated, no new tenant can be assigned to them without securely resetting the non-TEE nodes. The SC must act as the encryption proxy between the TEE and non-TEE nodes to ensure that all the data coming and leaving the non-TEE nodes are authenticated and encrypted.

From these observations, we conclude that to fully integrate TEE and non-TEE nodes into modern data centers, we need to carefully design the SC, the communication, and the attestation protocol to ensure strict security.

TABLE I: Distributed TEE challenges listed in Sec. III-D and how they are addressed in our design described in Sec. IV.

Distributed TEE challenges	Mitigation mechanisms
Malicious reconfiguration	SC + DSA Attestation (Sec. IV-C)
Persistent safe-state	Exclusive control plane access (Sec. IV-B)
Bypassing SC	Interception of all traffic in SC (Sec. IV-B)
Physical attacker	Encrypted traffic (Sec. IV-B1)
Scale-out	our scaling evaluation (Tables IV and V)

C. Overview of Hybrid Architecture (S_4) in Data Centers

In this section, we briefly overview our proposal, which adopts S_4 and realizes distributed heterogeneous enclaves running on CPUs, TEE-capable DSAs, and non-TEE devices. Our proposed system is built on top of existing data center architecture by keeping the cloud service provider's management plane (CSP-MP) unchanged. We assume that MP, software stack (OS, hypervisors, other applications), and data center infrastructure (NIC, switches, interfaces) are attacker-controlled. The CSP-MP is responsible for resource allocation/deallocation, load balancing, workload deployment, scaling, management, etc. All nodes have a hardware root-of-trust that can bootstrap the node securely. For TEE-capable devices, this root of trust is used for remote attestation to ensure the integrity of the firmware and the enclaves before the remote users provision secrets to the enclaves. On the TEE-capable nodes, the security monitor or SM, a high-privileged trusted entity on the device, enforces certain security proprieties, isolation, and attestation of enclaves. Therefore, the TEE-capable nodes support multi-tenancy as these nodes can accommodate multiple mutually distrusting enclaves. For non-TEE devices, we need additional trusted hardware in the rack, such as a security controller (SC). The SC is a low-TCB hardened trusted entity that can be verified and mediates the trusted path from a tenant to a device. Enclaves come with a manifest describing the devices' allocation strategy along with the required security properties. Once the resources (TEE and non-TEE) are allocated, the hypervisors are made aware of such a device by setting up MMIO and DMA regions. The trusted path from the remote verifier to the nodes are established in hierarchical way. The remote verifier first establishes a key with the CPU TEE by remote attestation, then the CPU establishes secure channels to the associated TEE-capable DSAs. This ensures that all MMIO and DMA transactions are confidentiality and integrity-protected.

D. Challenges

Although the potential design S_4 seems intuitively optimal, careful security considerations are still required. While monitoring data on the bus and secure communication between nodes are necessary, they are insufficient to determine the local node's configuration state. We list five challenges that we must address

TABLE II: Security properties and mechanisms to address them.

Security properties	Mechanisms
Identity & Attestation	SC, CPU and DSA HW-root-of-trust
Secure network path and bus	Authenticated encryption
Host memory protection	Memory encryption and integrity [39, 40]
DSA memory protection	3D-stacked memory [41, 42], encryption [43, 44, 45]
Tampering SC & non-TEE nodes	Tamper-detecting MCUs [46, 47]

for a datacenter-scale distributed TEE. A summary of these challenges and references to their solutions are listed in Table I.

(**1**) *Malicious reconfiguration:* Modern DSAs are highly configurable: the CSP/hypervisor can configure the DSAs' firmware. The hypervisors allow pass-through access to DSAs and can reconfigure their partition size [8], e.g., GPU partitions (MIG) [38] on NVIDIA A100 and H100.

(**2**) *Persistent safe-state:* Correct firmware and configurations during boot-up do not prevent malicious OS/hypervisors from re-configuring a node or flashing its firmware later. So, both TEE and non-TEE DSAs *must* be initialized from a safe state and stay in a safe state. Thus, the user or the SC must ensure the node firmware integrity is maintained before and after secret provisioning.

(**3**) *Bypassing SC:* The SC is unaware of the DSAs' execution due to their asynchronous nature. Therefore, malicious devices can attempt to bypass the SC and directly access other tenants' data. Thus, the TEEs or the SC must mediate all communications between nodes.

(**4**) *Physical attacker:* We must protect the SC, TEE, non-TEE nodes, and all the communication channels from a physical adversary.

(**5**) *Scale-out:* The ability to scale out must not compromise security guarantees. For example, if the workload requires multiple AI accelerators, our proposed design should provide an easy avenue for scaling.

E. Insights

Measured boot ensures firmware and configuration integrity, which the SC must verify and communicate to the user prior to secret provisioning. Remote attestation ensures that the DSAs are running the correct user-provided kernels. TEE-level isolation ensures that an attacker cannot access other tenants' memories on the same node. For non-TEE nodes, the SC is aware of tenants' resource allocations. At runtime, SC ensures tenant isolation by performing bus-level access control to verify if the accessor is authorized to access the resource. The communication channel between the nodes is secure, preventing attackers from manipulating data on the bus.

Physical Attacker. We assume a malicious CSP with full physical access to all the nodes and data center infrastructure. Therefore, we require protection against a physical attacker to ensure that the attacker-controlled CPS can not compromise the integrity and confidentiality of data and programs. The individual components, such as the CPU, DSA, SC, and the communication channel between them, must provide physical attack protection. Table II provides a set of desired security properties and the corresponding mechanisms.

(**1**) *CPU.* Existing systems such as memory encryption and integrity protection defend against physical attackers.

(**2**) *DSA.* Integrated memory in the DSA-SoC makes physical attack difficult. For off-core memory, memory encryption and integrity protection on DSAs can thwart physical attackers.

(**3**) *Non-TEE nodes.* The absence of the above mechanisms makes protecting non-TEE nodes challenging. The SC and non-TEE nodes are secured in a physical container with commercially available MCUs that detect physical tampering through pressure and vibration sensors [46, 47, 48, 49]. Such a mechanism is used by HETEE[22] and is suitable for our rack topology.

(**4**) *Communication channel.* Authenticated-encrypted (AES-GCM) channels between TEE/non-TEE nodes and SCs prevent physical attackers as long as the communication endpoints are inside the CPU-enclave, DSA-enclave, or SC.

IV. Security Controller (SC)

Our design secures both TEE and non-TEE nodes. The user is required to allocate *at least one* CPU-TEE node. This is necessary as our design requires one primary CPU node where the end-user deploys their job and runs the primary workload software stack, such as PyTorch for AI/ML. This ensures that an attacker-controlled OS/hypervisor on the TEE node cannot compromise the security of the distributed enclave. It is also required that the non-TEE nodes are *not* multi-tenant as they lack isolation primitives within that node. Resource allocation (such as memory, storage capacity, number of computation cores, etc.) is *static* and is decided during the enclave setup. DSAs execute programs, known as *kernels* that have predetermined memory allocation and do not allow new memory allocation (e.g., `malloc`) while inside the kernel. The user can define the size for storage or memory based on the workload requirements. Throughout this paper, we use the term *enclave* to define isolated environments either on CPU (isolated programs or confidential VM) or DSAs that cannot be manipulated from attacker-controlled software stack such as OS or hypervisor. Therefore, a programmer statically defines the memory sizes of each remote enclave in the job manifest.

Setup. The user submits the number, size, and type of each enclave and non-TEE node in a manifest file (Sec. IV-C), along with the code to be executed on each enclave and non-TEE node to the CSP-MP. The CSP-MP allocates the requested resources and trusted hardware on the rack, sends challenges, and collects attestation reports from them. The user gets a combined attestation report with the nodes' configurations. After successful verification, the trusted hardware generates and provisions an enclave-specific channel encryption key to start the computation. In the following, we discuss how to compose a rack with both non-TEE and TEE nodes and how to secure them.

A. SC Architecture

SC is a dedicated hardware module that is part of every rack, connected to all the nodes, and placed in a container along with the non-TEE nodes connected via PCIe. The container is equipped with physical tamper-detecting micro-controller units (MCUs) to prevent the physical attacker from accessing the non-TEE nodes. SC has a hardware root-of-trust capable of performing memory

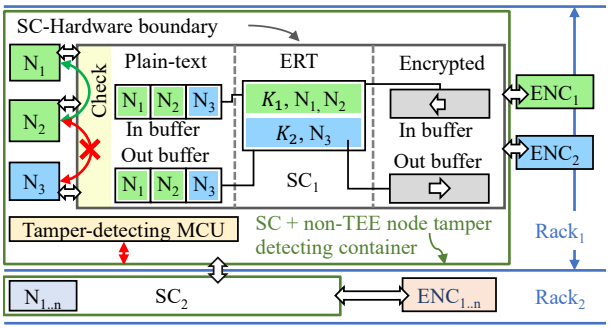

Fig. 3: **A high-level architecture of the SC** along with the non-TEE nodes (N_i) and TEE nodes (ENC_i). The SC and the non-TEE nodes are located in a tamper-detecting container with the help of a specialized microcontroller unit (MCU) [46, 47]. The enclave routing table (ERT) stores the key and corresponding nodes related to a distributed enclave workload. All these nodes and the SC container are located in a single data center rack. The figure also shows the communication link between two SCs, SC_1 and SC_2, which could be co-located in the same data center or across separate geographical locations.

isolation for itself. The SC's logic is implemented purely in the hardware (bare-metal) and acts as the *TEE proxy* for the non-TEE nodes and provides TEE properties such as isolation and a trusted path for them. It also ensures that a non-TEE node is allocated to a single tenant and is not shared with any other tenant or entity (e.g., CSP). The SC performs authenticated encryption for all communication between TEE and non-TEE memory using the shared secret ($\mathcal{K}$) provisioned during the initial attestation. For communication between non-TEE nodes, the SC does not perform authenticated encryption; instead, it enforces access control. The SC checks if a transaction's source and destination belong to the same job and, if so, allows it; otherwise, it blocks it. The SC has two sides for interacting with the nodes in the data center. The encrypted side is interfaced with the outside world (i.e., the TEE nodes and SCs in other racks). The plain text side is interfaced with the local non-TEE nodes confined within the tamper-proof container. Both sides have DRAM buffers that are DMA mappable to either TEE or non-TEE nodes to communicate with the SC.

At start-up, the SC first *resets* the non-TEE nodes to its factory configuration and creates locally unique identities (e.g., unforgeable PCIe physical *ids*) for them. The SC maintains input and output buffers for the non-TEE nodes. Since the data in these buffers are in plain text, the SC enforces isolated access to them. SC allocates non-overlapping regions of the buffers to individual non-TEE nodes, as shown in Fig. 3. This ensures that non-TEE nodes cannot access each other's plaintext data. To enforce access control between non-TEE nodes, the SC maintains a special data structure called an *Enclave Routing Table (ERT)*. The ERT keeps track of the nodes that are part of the same job. The SC adds entries to the ERT when one of the non-TEE nodes is allocated as part of a new job; the SC deletes the entry when either the job is completed or the CPU-enclave is disconnected or timed out.

B. Enforcing TEE Primitives

1) Setting up and reserving non-TEE nodes: The user of the distributed enclave submits an enclave manifest that describes the TEE and non-TEE resource requirements to the CSP-MP. An example enclave manifest is depicted in Fig. 4. As mentioned earlier, a job manifest must include a CPU-TEE node where all the primary software stacks (PyTorch, TensorFlow, Redis client, Database client, etc.) reside. In our design, such a CPU-TEE node could be a SEV-SNP-capable AMD CPU that can create confidential VMs. When CSP-MP allocates non-TEE nodes for a job, the SC first verifies if the nodes are unallocated by checking the ERT. If the nodes are available, the SC participates in the remote attestation process for the job. We describe the remote attestation process in detail in Sec. IV-C. On successful attestation, it programs a job-specific secret key $\mathcal{K}$ to the specific row of the ERT, as can be seen in Fig. 3 that shows two jobs with shared keys K_1, and K_2. After the remote attestation process, the CSP-MP can no longer tamper with the configuration of a distributed enclave. These shared symmetric keys are used to encrypt and authenticate (AES-GCM) all data between the SC and CPU-enclave. Such a setup mechanism ensures that scaling out the number of DSAs is independent of the host CPU. Therefore, our proposal can accommodate an increasing number of devices behind SC as long as the primary CPU-TEE can dispatch tasks to them.

2) Runtime integrity violation: The tamper-detecting MCU continuously monitors the physical integrity of the SC enclosure. The SC and all TEE nodes are reset upon detecting any violation, and any potential leakage is prevented. The SC is physically connected to the MCU and is part of the remote attestation for a job.

3) Communication: TEE and Non-TEE node: The SC provides staging buffers (incoming and outgoing buffers) where data can be encrypted and decrypted. For any data leaving the SC, the nodes move it to the SC's buffer, where it is encrypted with the destination's key before being sent out. Similarly, any data entering the SC (i.e., to one of the non-TEE nodes) arrives in the node's memory mapped in the SC, where it is decrypted with the destination key. Fig. 3 shows the details of this mechanism. Every message that arrives at the encrypted side (incoming buffer) contains the data, source, and destination node ID. The messages are encrypted and authenticated and can be decrypted and verified by the shared key in the ERT. The connection between the TEE nodes and the SC can be over PCI Express or Ethernet (e.g., RDMA over Converged Ethernet - RoCE). For both, we encrypt all the packets with the shared key between the CPU-enclave and SC established during the initial remote attestation. We assume the primary workload software stack, such as PyTorch for AI or the Redis client for in-memory databases, runs on the CPU-enclave (such as an AMD SEV-SNP confidential-VM).

Typically, the DSAs are memory mapped, i.e., the virtual address space of the DSA memory is mapped to the host memory address space. In this way, the host CPU can access the DSA memory as if it is part of its own memory. From the TEE node to a non-TEE DSA node behind the SC, the communication goes through the SC and gets decrypted or encrypted based on the direction of the communication. A small change in the device driver is required to ensure that all the DMA messages are intercepted and copied to the SC's incoming buffer when a CPU-TEE sends data to a non-TEE DSA over SC. Similarly, the SC's outgoing buffer is mapped to the CPU-TEE's memory,

where the enclave on the CPU node can access the encrypted data.

As a demonstration, we have three use cases to show the feasibility of communication over the SC: accessing large storage, in-memory databases, and AI. We provide more details of our experimental set-up in Sec. VI-C. The CPU-TEE communicates with the SSD and NPU (AI accelerator) over the SC for storage and AI use cases. SSD storage address paces and NPU memory space are DMA mapped to the SC's incoming and outgoing buffer at the plan text side (refer to Fig. 3). The SC's incoming and outgoing buffers on the encrypted side are mapped to the CPU-TEE. The SEV-VM copies the data (either file system data for storage workload or tensor/AI model for AI workload) to the incoming buffer on the SC's encrypted side, which is then decrypted by the SC and placed on the incoming buffer on the SC's plain text side. Similarly, the data from the DSA is copied to the out buffer on SC's plain text side, which is then encrypted and placed in the out buffer on SC's encrypted side, which is mapped to the CPU-TEE.

4) Communication: Non-TEE nodes behind SC: Unlike the communication between the TEE nodes, where the SC uses encryption to isolate traffic from the cloud provider, the SC uses the ERT to enforce the *access control* between the non-TEE nodes. The nodes communicate using the SC as an intermediary for performing data copies via an isolated staging buffer in the SC, where the non-overlapped part of the buffer is DMA mapped to the nodes. If the non-TEE nodes are allocated to the same enclave manifest, they share the same rows in the ERT. The ERT entries ensure that only the nodes of the same job (e.g., N_1 and N_2 in Fig. 3) can set up shared DMA buffers on SC. Once configured, the nodes can communicate over the shared DMA region.

5) Communication: Non-TEE nodes behind different SCs: If a job requires nodes distributed across multiple SCs, the local SC communicates with a remote SC over a secure channel using the job-specific key $\mathcal{K}$ to encrypt and decrypt the data. This is feasible since both SCs receive $\mathcal{K}$ during the setup and have valid ERT entries. Keeping a single key for the local and remote SC ensures that data encryption and decryption occur only once while communicating between a TEE and a non-TEE node. The local SC forwards all the encrypted data from the TEE node to the remote SC, where it is decrypted. Similarly, the data from a non-TEE node gets encrypted by the remote SC and forwarded to the TEE node via the local SC. Note that SCs do not have to synchronize ERT entries or key material.

However, the key difference is communicating between two non-TEE nodes across two different SCs. Typically, the communication between two non-TEE nodes behind the same SC does not involve any cryptographic operation, only access control. However, in this case, the data from a non-TEE node will be encrypted and authenticated before sending it to the remote SC, as the malicious CSP can observe and manipulate the data between SCs.

6) Releasing node: To terminate a job, the tenant's primary CPU enclave sends a termination command to the SC. The SC power resets the non-TEE nodes in that specific job and removes the mapping from ERT. Once reset, the SC notifies the CSP-MP that the node is available for future allocations.

7) Connecting TEE nodes: Unlike non-TEE nodes, TEE nodes do not depend on an SC as the local SM sets up the enclaves for tenants, and on-device TEE primitives ensure that the enclaves on the same node cannot access each other's memory. We use the hardware root of trust and remote attestation of the TEE nodes to ensure that such isolation is trustworthy. Globally, we use authenticated encryption to protect data leaving the enclave as shown in previous work [11]. Two CPU enclaves can communicate by establishing a key by mutual remote attestation and using this key to encrypt and authenticate all data. Such communication can be done over the socket interface. Communication between a CPU enclave and a DSA enclave can be achieved by existing approaches such as GPU [11, 50], NPU [51], FPGA [14], etc, with different confidential VM such as AMD SEV [32], ARM CCA [52, 53], Intel TDX [29].

```
{ "Job": "J1", "Version": "X.YZ",
  "Public Key": 0x12ff..,
  "TEE-Resource":[
    {"Type": "CPU", "Cores": 4, "Memory": "64G"},
    {"Type": "NPU", "Cores": 8, "Memory": "32G"}]
  "Non-TEE-Resource":[
{"Type": "SSD", "Capacity": "2T"}]
```

Fig. 4: **Example manifest file** where the distributed enclave developer specifies the number, size, and type of resource requirement for both TEE and non-TEE nodes.

C. Remote Attestation

Enclaves distributed over TEE and SC-backed non-TEE nodes require distributed attestation. Therefore, we ensure that the individual attestation reports from the nodes (code and configuration) and the combination of attestation reports are correct.

1) Enclave Manifest: A manifest in Figure 4 describes a user job where the developer specifies the number, size, and type of each CPU/DSA enclave and non-TEE node alongside the code to be executed on each enclave and non-TEE node. In the example manifest (Figure 4), the developer allocates both TEE and non-TEE nodes. The CPU and neural processing unit (NPU) are TEE-capable devices, whereas the SSD is non-TEE. For all these resources, the developer also specifies the amount of cores, memory, or capacity requirements. The remote verifier uses the manifest to verify the combined attestation report, which ensures that the allocations of the enclaves and non-TEE node are correct.

2) Attestation Process: The SC initiates the remote attestation. Given an enclave manifest, the SC first gets an allocation of the resources (physical nodes) from the CSP-MP. Then SC produces the attestation report for the CPU, DSA enclaves, and non-TEE nodes as follows:

SC Attestation. The SC generates a measurement of its firmware and version number signed with its root key to show the SC's authenticity. If the job requires non-TEE nodes across different racks, the primary SC performs the attestation by communicating with the SCs in each racks and performing mutual attestation. Thus, the primary SC can gather the attestation report of the nodes from other racks from the corresponding rack SC.

CPU-enclave Attestation. The SC uses the existing attestation mechanism supported on the CPU to collect the enclave attestation reports.

DSA-enclave Configuration Attestation. Each DSA has hardware support for generating an attestation report that includes configuration information and signing it with its root key. The SC simply collects the attestation reports.

Attestation of non-TEE Nodes. The attestation process involves resetting the non-TEE nodes to factory configurations and destroying any configurations from the previous tenant. Since non-TEE nodes do not have hardware support for this, the SC power-resets the device over a physical interface, forcing it to load factory configurations and firmware. The MCU responsible for detecting physical tampering [47] of the SC and non-TEE nodes, provides proof of the MCU secure boot and continuous monitoring of the physical tampering [46, 54]. The SC appends this proof to the combined attestation report for the job.

End to End Attestation flow. After receiving the enclave manifest from the user, the SC communicates with the corresponding CPU and DSA enclaves as described above and collects the attestation reports—therefore solving the *attestation challenge* of aggregating distributed attestation reports. The SC then sends the aggregate report to the remote verifier. The verifier checks the verification report of the primary SC to ensure that it is communicating with a genuine SC. Only then does the verifier check individual reports using hardware-specific attestation verification and ensure that the provisioning was done according to the manifest. Then, the verifier and the SC set up a shared key ($\mathcal{K}$) over a secure channel. Finally, the SC populates the ERT entry with the shared key and communicates it to the TEE nodes over a secure channel based on individual attestation reports.

D. Resource Allocation Strategy

We only expect the enclave resources to be static, typically for larger workloads in the cloud (e.g., an AWS F1 instance has 1 VM and 1 FPGA). This is also reflected in the enclave manifest (Fig. 4) that dictates the resource requirement of the distributed enclave. Our current design allows the cloud provider to manage dynamic resource management for non-enclave nodes not committed to the enclave. We can add support for dynamic scaling of enclave resources (e.g., attaching a new GPU); this will require a new attestation, after which we can update the ERT.

V. Security Analysis

We analyze the security of our design in the presence of different adversary capabilities.

A. Attack from Untrusted CSP-MP

CSP-MP can violate the job manifest by allocating the wrong enclaves and non-TEE nodes, potentially with outdated and possibly compromised firmware. The remote verifier will detect such violations via the attestation report from enclaves. TEE nodes' local memory isolation prevents the CSP-MP or hypervisor from accessing/modifying the enclave's private memory. Authenticated encryption of data leaving a trusted enclave prevents compromised physical devices, e.g., switches added by CSP-MP, from intercepting traffic, triggering rogue DMA/RDMA, and injecting interrupts into nodes. Enclaves' local SMs prevent CSP-MP from allocating the same enclave to two different tenants or jobs. The SC locks a non-TEE node to a job (using the ERT) during the remote attestation. Authenticated encryption of data between SC across different racks also prevents malicious CSP from accessing and/or manipulating the data.

B. Malicious Configurations

Untrusted hypervisors or CSP-MP can push malicious configurations to only the TEE nodes, where the remote verifier can verify the nodes' SM configurations and their integrity via the attestation report. For non-TEE nodes, the SC resets them to their safe factory configuration before allocating them to an enclave.

C. Attacks via Non-TEE Nodes

All the access from the non-TEE node goes through SC's ERT to prevent a malicious non-TEE node from accessing SC's buffer and reading other nodes' plain text data. ERT also prevents non-TEE nodes from communicating with each other (over MMIO/DMA) unless the enclave manifest explicitly allows communication. ERT stores all keys used to encrypt data leaving the non-TEE nodes. Malicious non-TEE nodes cannot read out the ERT, tamper with the ERT, or compromise the SC's execution as SC is isolated and inaccessible to all other entities. The non-TEE nodes can try to send malicious MMIO/DMA requests to non-TEE nodes behind other SCs or other TEE enclaves. This does not compromise confidentiality and integrity, as all data is protected with AES-GCM.

D. Physical Attacker

The tamper-resilient MCU on the SC container prevents the attacker from probing non-TEE nodes; communications between non-TEE nodes and the SC never leave the SC boundary. Power resetting SC and non-TEE nodes will destroy their internal states and not leak any data. The TEE node's memory encryption and integrity protection safeguard memory. A physical attacker cannot see or modify authenticated-encrypted DMA data between TEE nodes or messages between non-TEE nodes from different SCs. The physical attacker cannot leak data from the previous tenant by disconnecting TEE nodes from the SC and allocating non-TEE nodes, as all data leaving the SC is protected with authenticated encryption.

E. Security against Side-channels

Our solution does not prevent existing side channels, such as the one present in the CPU or GPU micro-architecture. Our design cannot prevent side channels from higher-level applications, e.g., secret-dependent communication between nodes. However, we will emphasize that SC can be combined with existing side channel mitigation techniques to reduce the attack surface. Therefore, addressing the side channels is out of the scope of this paper.

VI. Implementation and Evaluation

Evaluation Setup. Table III describes the hardware platforms, OS, runtime, and drivers used in our evaluation. Our evaluation has four components: the CPU TEE, the synthesized SC hardware, an AI accelerator (NPU) that serves as a non-TEE device, and cycle-accurate simulation of SSDs that serves as both TEE and non-TEE nodes. In the following, we summarize

TABLE III: Evaluation platforms used in our SC experiments.

Evaluation	Hardware	OS/RT/Driver
CPU TEE	AMD CPU with SEV-SNP, 256GB DRAM	Ubuntu 20.04 + KVM
SC	870MHz 28 nm CMOS ASIC	-
AI acc	Huawei Ascend 910A NPU, 32 core	Ascend driver + runtime
SSD sim	Intel CPU, 128 GB DRAM	Ubuntu 20.04 + DRAMSim3

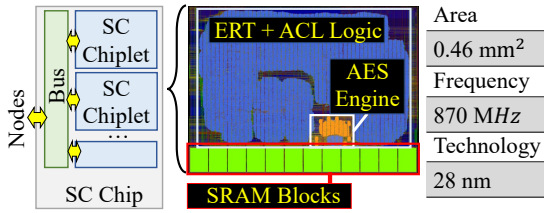

Fig. 5: **Floor plan** of the synthesized SC chiplet with ERT, SRAM block, and the AES-GCM engine. The figure shows a high-level SC chip design with individual SC chips connected over a fast optical bus. The bus also connects the SC to the TEE and non-TEE nodes.

our implementation of the Security Controller and present our evaluation findings.

A. Security Controller (SC) implementation

The SC prototype has two units: the control and the data plane. The control plane is responsible for adding/removing nodes, remote attestation of enclaves, and spawning VMs, and is implemented on an FPGA using the Vitis cryptography library [55]. The data plane is responsible for data streams, cryptographic operation, and access control (Fig. 3) and is implemented and synthesized as an ASIC. Due to performance criticality, we implement the *SC chiplet* data plane prototype in RTL (~ 2.5 KLoC of SystemVerilog code). We synthesize our SystemVerilog RTL implementation on a high-*k* metal gate (HKMG) 28 nm CMOS technology, corresponding to a multi-VT standard cell library (supply voltage of 0.8 V, estimated in typical-typical (TT) corner). Multiple chiplets are connected over an optical demultiplexing-based bus. We only synthesize the SC chiplet as the demultiplexer unit on the bus is a complex analog IP, and an existing design has near-zero demultiplexing latency (12.5 ps) [56]. Fig. 5 shows a synthesized SC prototype chiplet that comprises one AES-GCM engine, 64 KB (2 × 32 KB for in and out buffer, ref Fig. 3) SRAM, and the ERT. The ERT is a 256-bit wide hardware lookup table with 1024 node entries. Our prototype can be interfaced with the memory controller and DMA engine to DMA data in and out of the SC.

B. SSD TEE Node

Simulator Base Model. The base SSD simulation is based on SimpleSSD [57], a state-of-the-art cycle-accurate full-system simulator that closely simulates commercial SATA and NVMe SSDs. SimpleSSD provides two simulation modes: i) standalone for quick prototyping on the SSD hardware and firmware model and experimenting with trace-based workloads, and ii) full-system simulation based on Gem5, a popular CPU micro-architectural research platform. However, the latter is only used for functional validation due to the slow (a few KIPS) simulation rate. We evaluated the SSD workloads in the standalone mode

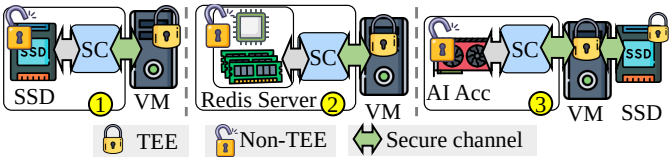

Fig. 6: **Three end-to-end workload evaluations with SC.**

for performance and used the full-system simulation to ensure that our modifications are valid and can boot Linux correctly. SimpleSSD simulates all the IO and management operations and provides a block storage device to boot Linux and execute IO operations. It computes the timing of each IO and management operation separately based on the timings of the memory chip (DRAM and NAND flash) and ISA (CPI of the SSD controller core).

SSD Hardware Changes. To enable SSD-enclave, we implement several hardware changes on the base model. An SSD-enclave comprises a contiguous NAND physical address range known as *namespace* and the isolated DRAM cache to accelerate read-write operations. We implement a hardware look-up table (LUT) that contains the mapping between the namespace id, the associated CPU-enclave id, and the AES-GCM key for authenticated encryption. This LUT can only be accessed by the privileged SSD firmware where all the access control logic is implemented. The access control unit (ACU) is a hardware compactor module that checks the enclave or TEE VM ID in the DMA transaction and checks it with the LUT entries. Based on the comparison result, ACU permits or rejects DMA operations. As all the hardware checks are done before the IO operation hits the DRAM cache and all page translations, our mechanism does not require any changes in the SSD controller; therefore, the SSDs' performance characteristics remain unchanged. All the DMA transactions are encrypted/decrypted by AES-GCM implemented on the SSD firmware. To implement Remote attestation (RA), we use an off-the-shelf chip for secure provisioning and measured boot. We store a root key in the RA to emulate the manufacturer's key provisioning. During startup, the RA executes a measured boot of the SSD firmware image. The measured boot results from the PCR banks are signed with the root key and used as a measurement report.

C. Cloud-Centric Workload Design

We evaluate our design on three different workloads that we design that are representative of cloud deployments.

Evaluation Workloads. We conduct three experiments with real-world cloud-focused workloads using an AMD SEV VM as the CPU enclave shown in Fig. 6. The SEV VM is the primary host node that the remote user accesses (e.g., over `ssh`). These three different workloads stress different devices and different parts of the system, as described in the following:

① *Storage intensive workload.* The SEV VM accesses a file system on a non-TEE SSD node behind SC. This emulates storage-intensive workloads such as database accesses, reading large model files from SSDs for AI/ML, etc.

② *In-memory database workload.* The SEV VM accesses a Redis server, a large in-memory database server that serves read

and update queries on in-memory objects. The Redis server is a non-TEE x86 node behind the SC.

③ *AI/ML workload.* The SEV VM gets inference data from an SSD that provides DMA encryption (via a hardware AES-GCM engine that encrypts all DMA traffic between the host and the SSD) and runs inference workload on a non-TEE NPU: Atlas 200 DK [58](2 core Ascend 310 SoC + ARMv8 host), behind SC.

Currently, direct PCIe devices (Ascend NPU) are incompatible with AMD SEV VMs due to encrypted DMA addresses [59, 60] that the devices are unable to access; however, SSDs are compatible with `VirtIO`.

Evaluation Methodology. We run these three experiments as the following:

① We use the FIO [61], a well-known Linux-based file benchmark. We evaluate with six FIO workloads (refer to Fig. 7) over 4K block size. FIO on the SEV VM produces two IO traces: submission and completion, which show when the IO commands are submitted and completed. The submission traces are submitted to the synthesized SC to account for the cost of AES-GCM and access control check. The output traces from the SC are then used by the SSD simulator. The resulting trace is then sent back to the SEV VM via the SC, compared to the completion trace as the baseline.

② We use Yahoo! Cloud Serving Benchmark [62] to run Redis on different database configurations (R: record size, and O: number of read/update operations). The Redis client running on an AMD SEV VM generates the IO traces (database query and submission time), which are first encrypted and then sent to the SC for decryption and access control check. Then, SC passes it to a Redis server, and the resulting data is sent to the SEV-VM through SC.

③ The ImageNet data sets are read from our TEE-enabled SSD simulator from the SEV VM and sent to a non-TEE Atlas 200 DK to run Resnet-34 and Resnet-50 behind SC. The inference results are then sent back to the VM over SC. We use ResNet-34 and ResNet-50 with batch sizes 1 and 16.

D. SC's Effect on Workloads

The synthesized SC prototype runs at 870 MHz, and its end-to-end latency is 1.15 ns. A single SC chiplet transfers 16 Bytes/cycles, giving it a practical throughput of 12.96 GB/s. For comparison, 100GbE Ethernet has a theoretical throughput of 12.5 GB/s.

SC Scaling Performance. Tables IV and V show the SC scaling for real-world AI inference workloads running on Ascend nodes and FIO benchmark on SSDs, respectively. For ResNet-34 with batch size = 1, the AI core takes 1.24 ms/image and can be saturated with a data rate of 133.88 MB/s from a CPU-enclave, assuming an average 170 KB JPEG image in the ImageNet dataset. Therefore, a single SC chiplet can handle concurrent 90 CPU-enclaves streaming image data. However, the number of connected nodes is limited by the available PCIe lanes. Adding jobs requires sending the pre-trained models to the NPU, which is expensive but a one-time cost. A single SC chiplet can add 31.85 jobs/sec of VGG-16, the largest model we evaluated (380 MB), without any performance degradation. For SSDs, we ran sequential and random read-write FIO benchmarks and observed up to 117 concurrent Intel 535 SSDs.

TABLE IV: **SC Scaling with AI workloads on Ascend 910 NPU.** Speed: single image inference time (ms) for a given batch size (B) and resolution (Res), saturation size ($\mathcal{S}$): data rate (MB/s) from a CPU-enclave that saturates an AI core (using ImageNet dataset). Based on $\mathcal{S}$, we calculate the number of concurrent Ascend NPU nodes ($\mathcal{N}$) that a *single* SC chiplet can serve.

AI models	B/Res	Speed/$\mathcal{S}$	$\mathcal{N}$	AI models	B/Res	Speed/$\mathcal{S}$	$\mathcal{N}$
ResNet-34	1/224	1.24/133.88	90	RetinaNet	1/800	30.68/5.41	2236
	8/224	6.13/215.6	56	VGG-16	1/300	6.63/25.04	483
	16/224	11.82/224.72	53	UNet	1/572	23.48/7.07	171
ResNet-50	1/224	1.91/86.91	139	YOLOV3	1/256	3.75/11.27	273
	8/224	10.87/122.18	99		1/416	7.71/21.53	562
	16/224	24.67/107.67	112		8/256	19.45/68.28	177

TABLE V: **SC Scaling with SSD workload** for SSDs running FIO benchmarks sequential read/write (S-RW), and random read/write (R-RW) with speed (MB/s) and number of concurrent SSD nodes ($\mathcal{N}$).

SSD	S-RW		R-RW	
	Speed	$\mathcal{N}$	Speed	$\mathcal{N}$
Intel 700 400 G	362.75	33	335.1	36
Samsung Z-SSD 400G	394.00	30	383.6	31
Samsung 983DCT 1.92T	352.02	34	355.7	34
Samsung 850Pro 256G	286.48	42	279.8	43
Intel 535 240G	120.23	100	102.9	117

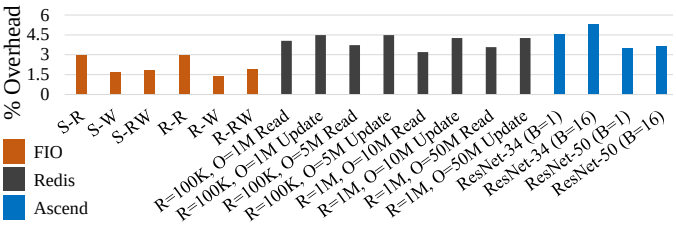

Fig. 7: **Overhead of end-to-end SC evaluations** The figure shows the overhead of three SC evaluations as depicted in Fig. 6.

Initialization and Attestation. We evaluate the attestation of SC, TEE, and non-TEE nodes from the SC-FPGA prototype. The CPU-enclave are implemented as secure-boot-capable Fedora VMs running on KVM. Upon receiving an enclave manifest (in a KVM `dumpxml` format) from the SC, the hypervisor on the CPU node creates a new VM. After the confidential-VMs boot, SC collects the signed secure boot traces (~245 KB) from the SW-TPM of the VMs and DSAs. For key exchange, the SC and the nodes use the well-known SIGMA protocol [63].

End-to-end Workload Evaluation Results. The results are shown in Fig. 7 for three end-to-end SC evaluations in Sec. VI-C.

① *FIO Benchmark:* In the file system experiment involving the FIO bechmarking tool, for a single sequential read-write and random read-write over 4K blocks, SC and AES-GCM in the SEV VM introduce latency overheads of 1.84% and 1.94%, respectively. However, IO requests are queued from the CPU and dispatched in bursts and are 10.237 µs and 6.607 µs apart for sequential and random read-write benchmarks. A single 4K block incurs 3.585 µs latency at the SC for access control and AES-GCM. Therefore, 8 IO requests (for a 32K SC buffer) can be pipelined to a single SC chip to hide the latency completely.

② *Redis:* We evaluated the effect of SC on read and update query latency in three sets of parameters. We use the default record size, which is 1KB. We ensure that the number of operations (O) is large enough, i.e., >10× the record size (R) to cover all records. We use the default Zipfian distribution to generate the IO requests. We observe that, on average, SC introduces 3.6% and 4.37% overhead in the read and update query latency.

③ *AI Workload:* In the combined evaluation with TEE-enabled SSD and non-TEE NPU, we use both devices without SC as the baseline. Then, we calculate the additional latency introduced by the TEE SSD, SEV VM's AES-GCM, and SC's operation. Such a low overhead is expected, as the NPU's inference time is dominant. E.g., we observe a maximum of 5.3% overhead for ResNet-34 with batch size 16, where the end-to-end timing is dominated by the NPU (11.76 ms) compared to the SSD (87.04µs) and SC (11.28µs).

VII. Related Work

Existing solutions [64, 65] deployed in production assume a trusted CSP and only protect against a software attacker executing as a malicious tenant. In contrast, we focus on a stronger adversary model: an untrusted CSP and a physical attacker.

Device TEEs. Prior works that enable TEEs on DSAs such as GPU [11, 66, 50, 67, 68], FPGA [14, 69, 70, 71], NPU [72], and in-storage execution [13] focus solely on creating enclaves on the DSAs isolated from the untrusted software. Some DSA-TEEs additionally protect against a physical adversary. The DSAs are connected to a single physical host that may be TEE-capable. The only known commercially available DSA-TEE is NVIDIA's H100 [12], which enables in-GPU isolated instances (MIG [38]) using proprietary GPU firmware. Additionally, prior works [73, 74, 75, 76] use cryptographic primitives for the cloud.

Heterogeneous TEE. SGX-FPGA [77] proposes a custom FPGA enclave connected to an SGX enclave over an encrypted channel. Similarly, HIX [50] enables shared memory communication between a GPU enclave and an SGX enclave using HIX trusted driver. In Hector-V [78], during boot time, CPU-SM configures an access control for peripherals on the AXI4-lite bus to drop requests coming from untrusted code. vTEE [79] uses micro kernel and p2p network to connect heterogeneous device. Existing proposals [19, 20, 21] extend CPU-TEE protection by leveraging MMU to protect MMIO devices and perform bus-level isolation since devices are physically connected to a TEE host. Our solution is compatible with these device-TEEs but does not require the devices to be directly connected to CPU-TEE hosts. We use the insights from the above proposals to build the TEE-support for AI accelerator and SSD by adding protection units (ACU, FMT, MPE) and relying on security monitors. Our SC-based access control for distributed settings over a network mimics the bus-level filtering for a centralized setting where the CPU and devices are connected to a bus.

Distributed TEE. HETEE [22] assumes non-TEE devices and hosts are connected to a centralized security controller over a PCIe switch in a physical attack-proof container, limiting its scalability to a single rack and ~ 60 nodes. In contrast, we carefully avoid a centralized SC design that causes bottlenecks in HETEE. Unlike HETEE, we allow multi-tenancy on TEE nodes. Furthermore, the SC has to perform AES-GCM for all data between 60 nodes. Hence, they employ trusted proxy nodes to distribute the cryptographic operations evenly. We avoid this problem by protecting the non-TEE nodes with the SC's access-control checks, and AES-GCM is only required when communicating with TEE nodes.

Commercial Confidential Cloud Solutions. Several CSPs offer accelerators as a unit [80, 81] or as a packaged service [82, 83]. Many cloud provider provides CPU-based TEEs based on Intel SGX, Intel TDX, and AMD SEV [15, 16, 17, 84], and hypervisor [18].

Trusted IO. Upcoming PCIe features enable mechanisms to connect CPU-TEEs with DSA-TEEs. Specifically, TEE Device Interface Secure Protocol (TDISP) for PCIe-6 enables TEEs on processors to connect to TEE-enabled PCIe DSAs [85]. Integrity and Data Encryption (IDE) on PCIe-5 encrypts, and integrity protects PCIe traffic on processors and DSAs [86]. The adoption of these PCIe extensions as TDX-Connect for Intel TDX, SEV-TIO for AMD SEV-SNP, Device Assignment (DA) for Arm CCA, and IOPMP for RISC-V allows these TEE-enabled DSAs secure direct access to TEE memory on processors [87, 32, 88, 89]. These technologies allow a single CPU node to connect to a TEE-enabled DSA node securely. For example, with SEV-TIO, a TEE-enabled DSA connected to an SEV-SNP VM can directly access the VM's memory. Here, PCIe IDE encrypts, and integrity protects all traffic between the SEV-SNP VM and the DSA, eliminating the need for the bounce-buffer-style software encryption we propose. However, this is only feasible if the DSA is directly connected to the SEV node. Therefore, we need to address the challenges in a distributed setting that we consider. Single-node performance will improve when nodes with these technologies are available in data centers. However, the security principles and insights we outline in this paper will still be required to deploy these technologies securely at the datacenter scale. CXL and all its variants, CXL-mem, CXL-cache, and CXL-IO, similar to PCI-E, is *single root*, i.e., there can be only one host controller. In contrast, our proposal allows arbitrary scaling with the help of SC. CXL has similar security mechanisms as the PCI-E IDE standard, i.e., end-to-end encrypted transmission between hubs [90], which trusted motherboard/chipset manufacturers can enable. Detailed security analysis and design with CXL support is beyond the scope of our paper and requires independent exploration.

VIII. Conclusion

We propose a data center design with enclaved execution guarantees for TEE and non-TEE devices, scaling across nodes and future-ready for TEE devices. We show that the performance costs of such a design are modest and can be reduced further with fine-tuning. We envision that our insight will serve as guiding principles for cloud-scale solutions for confidential computing beyond CPU execution.

References

[1] NVIDIA, "Nvidia A100 GPUs power the Modern Data Center."

[2] S. Yesil, M. M. Ozdal, T. Kim, A. Ayupov, S. Burns, and O. Ozturk, "Hardware accelerator design for data centers," in *IEEE/ACM ICCAD*, 2015.

[3] T. J. Ham, L. Wu, N. Sundaram, N. Satish, and M. Martonosi, "Graphicionado: A high-performance and energy-efficient accelerator for graph analytics," in *IEEE MICRO*, 2016.

[4] run.ai, "Keras Multi GPU: A Practical Guide."

[5] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A Computation-Centric distributed graph processing system," in *USENIX OSDI*, 2016.

[6] M. Lavasani, H. Angepat, and D. Chiou, "An fpga-based in-line accelerator for memcached," *IEEE Computer Architecture Letters*, vol. 13, no. 2, pp. 57–60, 2013.

[7] H. AlJahdali, A. Albatli, P. Garraghan, P. Townend, L. Lau, and J. Xu, "Multi-tenancy in cloud computing," in *IEEE SOSE*, 2014.

[8] M. Adler, K. E. Fleming, A. Parashar, M. Pellauer, and J. Emer, "Leap scratchpads: automatic memory and cache management for reconfigurable logic," in *ACM/SIGDA FPGA*, 2011.

[9] Intel, "Intel software guard extensions," https://software.intel.com/content/www/us/en/develop/topics/software-guard-extensions.html.

[10] AMD, "AMD SEV-SNP," https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf.

[11] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on gpus," in *USENIX OSDI*, 2018.

[12] "NVIDIA Hopper Architecture In-Depth," https://developer.nvidia.com/blog/nvidia-hopper-architecture-in-depth/, 2022.

[13] L. Kang, Y. Xue, W. Jia, X. Wang, J. Kim, C. Youn, M. J. Kang, H. J. Lim, B. Jacob, and J. Huang, "Iceclave: A trusted execution environment for in-storage computing," in *IEEE/ACM MICRO*, 2021.

[14] M. Zhao, M. Gao, and C. Kozyrakis, "Shef: shielded enclaves for cloud fpgas," in *ACM ASPLOS*, 2022.

[15] Microsoft, "Azure confidential cloud - protect data in use — microsoft azure," https://azure.microsoft.com/en-us/solutions/confidential-compute/.

[16] Google, "Confidential Computing — Google Cloud."

[17] IBM, "Confidential computing for total privacy assurance – IBM."

[18] Amazon, "Aws nitro enclaves - create additional isolation to further protect highly sensitive data within ec2 instances," https://aws.amazon.com/ec2/nitro/nitro-enclaves/.

[19] M. Schneider, A. Dhar, I. Puddu, K. Kostiainen, and S. Čapkun, "Composite enclaves: Towards disaggregated trusted execution," *IACR CHES*, 2022.

[20] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "CURE: A security architecture with customizable and resilient enclaves," in *USENIX Security 21*, 2021.

[21] J. Jiang, J. Qi, T. Shen, X. Chen, , S. Zhao, S. Wang, L. Chen, N. Zhang, X. Luo, and H. Cui, "Cronus: Fault-isolated, secure and high-performance heterogeneous computing for trusted execution environments," in *ACM/IEEE Micro*, 2022.

[22] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, B. Zhao, Z. Wang, Y. Zhang, J. Ying, L. Zhang *et al.*, "Enabling rack-scale confidential computing using heterogeneous trusted execution environment," in *IEEE S&P*, 2020.

[23] N. Alachiotis, A. Andronikakis, O. Papadakis, D. Theodoropoulos, D. Pnevmatikatos, D. Syrivelis, A. Reale, K. Katrinis, G. Zervas, V. Mishra *et al.*, "dredbox: A disaggregated architectural perspective for data centers," in *Hardware Accelerators in Data Centers*. Springer, 2019, pp. 35–56.

[24] S. Legtchenko, H. Williams, K. Razavi, A. Donnelly, R. Black, A. Douglas, N. Cheriere, D. Fryer, K. Mast, A. D. Brown *et al.*, "Understanding Rack-Scale disaggregated storage," in *HotStorage*, 2017.

[25] Fungible, "PCIe device disaggregation: Fungible, enabling hyperdisaggregation of compute and storage resources across data center scales," https://www.fungible.com/technology-showcase/pcie-device-disaggregation/, 2021.

[26] K. Lim, J. Chang, T. Mudge, P. Ranganathan, S. K. Reinhardt, and T. F. Wenisch, "Disaggregated memory for expansion and sharing in blade servers," *ACM SIGARCH computer architecture news*, vol. 37, no. 3, pp. 267–278, 2009.

[27] R. Lin, Y. Cheng, M. De Andrade, L. Wosinska, and J. Chen, "Disaggregated data centers: Challenges and trade-offs," *IEEE Communications Magazine*, vol. 58, no. 2, pp. 20–26, 2020.

[28] S.-Y. Tsai, Y. Shan, and Y. Zhang, "Disaggregating persistent memory and controlling them remotely: An exploration of passive disaggregated Key-Value stores," in *USENIX ATC*, 2020.

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

[30] ARM, "Arm confidential compute architecture (ARM-CCA)," https://www.arm.com/why-arm/architecture/security-features/arm-confidential-compute-architecture.

[31] V. Costan and S. Devadas, "Intel sgx explained." *IACR Cryptology ePrint Archive*, vol. 2016, no. 086, pp. 1–118, 2016.

[32] AMD, "AMD SEV-TIO: Trusted I/O for Secure Encrypted Virtualization," March 2023.

[33] M. Schneider, R. J. Masti, S. Shinde, S. Capkun, and R. Perez, "Sok: Hardware-supported trusted execution environments," *arXiv preprint arXiv:2205.12742*, 2022.

[34] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[35] N. S. Agency, "U.S. government protection profile for separation kernels in environments requiring high robustness," https://www.niap-ccevs.org/pp/ppskpphrv1.03.pdf, 2007.

[36] Intel, "PCI Express Device Security Enhancements - pcie-device-security-enhancements.pdf," 2018.

[37] S. Checkoway and H. Shacham, "Iago attacks: why the system call api is a bad untrusted rpc interface," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 253–264, 2013.

[38] Nvidia, "Nvidia multi-instance gpu user guide :: Nvidia tesla documentation," https://docs.nvidia.com/datacenter/tesla/mig-user-guide/.

[39] D. Kaplan, J. Powell, and T. Woller, "AMD memory encryption," *White paper*, 2016.

[40] Intel, "Runtime encryption of memory with intel® total memory encryption - multi-key," https://www.intel.com/content/dam/www/central-libraries/us/en/documents/2022-10/intel-total-memory-encryption-multi-key-whitepaper.pdf.

[41] AMD, "Amd instinct mi250x accelerator," https://www.amd.com/en/products/server-accelerators/instinct-mi250x, 2022.

[42] D. Foley, "Nvlink, pascal and stacked memory: Feeding the appetite for big data," https://developer.nvidia.com/blog/nvlink-pascal-stacked-memory-feeding-appetite-big-data/, 2014.

[43] S. Na, S. Lee, Y. Kim, J. Park, and J. Huh, "Common counters: Compressed encryption counters for secure gpu memory," in *IEEE HPCA*, 2021.

[44] S. Yuan, A. W. B. Yudha, Y. Solihin, and H. Zhou, "Analyzing secure memory architecture for gpus," in *IEEE ISPASS*, 2021.

[45] S. Lee, J. Kim, S. Na, J. Park, and J. Huh, "TNPU: Supporting trusted execution with tree-less integrity protection for neural processing unit," in *IEEE HPCA*, 2022.

[46] STMicroelectronics, "STMicroelectronics and Prove & Run Provide Scalable Security Platform for IoT Devices," 2017.

[47] ——, "STM32L4 series of ultra-low-power MCUs," 2022.

[48] B. Nisarga and E. Peeters, "System-level tamper protection using msp mcus," *Texas Instruments*, 2016.

[49] Southco, "RACK LEVEL SECURITY," 2022.

[50] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh, "Heterogeneous isolated execution for commodity gpus," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 455–468.

[51] K. Vaswani, S. Volos, C. Fournet, A. N. Diaz, K. Gordon, B. Vembu, S. Webster, D. Chisnall, S. Kulkarni, G. Cunningham *et al.*, "Confidential computing within an {AI} accelerator," in *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, 2023, pp. 501–518.

[52] S. Sridhara, A. Bertschi, B. Schlüter, M. Kuhne, F. Aliberti, and S. Shinde, "ACAI: Protecting accelerator execution with arm confidential computing architecture," in *33rd USENIX Security Symposium (USENIX Security 24)*, 2024, pp. 3423–3440.

[53] C. Wang, F. Zhang, Y. Deng, K. Leach, J. Cao, Z. Ning, S. Yan, and Z. He, "Cage: Complementing arm cca with gpu extensions." ISOC, 2024.

[54] STMicroelectronics, "Stsafe-a110: Authentication, state-of-the-art security for peripherals and iot devices," https://www.st.com/en/secure-mcus/stsafe-a110.html, 2022.

[55] Zilinx, "Vitis_libraries/security at main - xilinx/vitis_libraries - github," https://github.com/Xilinx/Vitis_Libraries/tree/main/security, 2022.

[56] C.-S. Brès, A. O. Wiberg, B.-P. Kuo, J. M. Chavez-Boggio, C. F. Marki, N. Alic, and S. Radic, "Single-gate 320-to-8× 40 gb/s demultiplexing," in *National Fiber Optic Engineers Conference*. Optical Society of America, 2009, p. PDPA4.

[57] CAMELab, "Simplessd - simplessd 2.0.12 documentation," https://docs.simplessd.org/en/v2.0.12/index.html.

[58] Huawei, "Atlas 200 dk ai developer kit — huawei global," https://e.huawei.com/in/products/cloud-computing-dc/atlas/atlas-200, 2022.

[59] Suse, "Amd secure encrypted virtualization (amd-sev) guide," https://documentation.suse.com/sles/15-SP1/html/SLES-amd-sev/index.html, 2022.

[60] R. Hat, "Chapter 12. Configuring AMD SEV Compute nodes to provide memory encryption for instances," 2022.

[61] J. Axboe, "fio - flexible i/o tester rev. 3.32," https://fio.readthedocs.io/en/latest/fio_doc.html, 2017.

[62] brianfrankcooper, "Github - brianfrankcooper/ycsb: Yahoo! cloud serving benchmark," https://github.com/brianfrankcooper/YCSB, 2022.

[63] H. Krawczyk, "Sigma: The'sign-and-mac'approach to authenticated diffie-hellman and its use in the ike-protocols." in *Crypto*, vol. 2729. Springer, 2003, pp. 400–425.

[64] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *SoCC*, 2011.

[65] R. Yu, G. Xue, V. T. Kilari, and X. Zhang, "Network function virtualization in the multi-tenant cloud," *IEEE Network*, vol. 29, no. 3, pp. 42–47, 2015.

[66] T. Hunt, Z. Jia, V. Miller, A. Szekely, Y. Hu, C. J. Rossbach, and E. Witchel, "Telekine: Secure computing with cloud GPUs," in *NSDI*, 2020.

[67] L. K. Ng, S. S. Chow, A. P. Woo, D. P. Wong, and Y. Zhao, "Goten: Gpu-outsourcing trusted execution of neural network training," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

[68] O. Kwon, Y. Kim, J. Huh, and H. Yoon, "Zerokernel: Secure context-isolated execution on commodity gpus," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 4, pp. 1974–1988, 2019.

[69] S. Zeitouni, J. Vliegen, T. Frassetto, D. Koch, A.-R. Sadeghi, and N. Mentens, "Trusted configuration in cloud fpgas," in *IEEE FCCM*, 2021.

[70] H. Oh, K. Nam, S. Jeon, Y. Cho, and Y. Paek, "Meetgo: A trusted execution environment for remote applications on fpga," *IEEE Access*, vol. 9, pp. 51 313–51 324, 2021.

[71] W. Ren, J. Pan, and D. Chen, "Accguard: Secure and trusted computation on remote fpga accelerators," in *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*. IEEE, 2021, pp. 378–383.

[72] W. Ren, W. Kozlowski, S. Koteshwara, M. Ye, H. Franke,

and D. Chen, "Accshield: a new trusted execution environment with machine-learning accelerators," in *ACM/IEEE Design Automation Conference*, 2023.
[73] A. López-Alt, E. Tromer, and V. Vaikuntanathan, "On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption," in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, 2012, pp. 1219–1234.
[74] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the gpu," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1021–1038.
[75] E. Hesamifard, H. Takabi, M. Ghasemi, and R. N. Wright, "Privacy-preserving machine learning as a service." *PETS*, 2018.
[76] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious Multi-Party machine learning on trusted processors," in *USENIX Security*, 2016.
[77] K. Xia, Y. Luo, X. Xu, and S. Wei, "Sgx-fpga: Trusted execution environment for cpu-fpga heterogeneous architecture," in *ACM/IEEE DAC*, 2021.
[78] P. Nasahl, R. Schilling, M. Werner, and S. Mangard, "Hector-v: A heterogeneous CPU architecture for a secure RISC-V execution environment," in *ACM Asia CCS*, 2021.
[79] A. Koshiba, F. Gust, J. Pritzi, A. Vahldiek-Oberwagner, N. Santos, and P. Bhatotia, "Trusted heterogeneous disaggregated architectures," in *Proceedings of the 14th ACM SIGOPS Asia-Pacific Workshop on Systems*, 2023, pp. 72–79.
[80] A. AWS, "Amazon ec2 f1 instances," https://aws.amazon.com/ec2/instance-types/f1/, 2022.
[81] Microsoft, "Deploy ml models to field-programmable gate arrays (fpgas) with azure machine learning," https://learn.microsoft.com/en-us/azure/machine-learning/v1/how-to-deploy-fpga-web-service, 2022.
[82] Huawei, "Modelarts - train ml models || huawei cloud," https://www.huaweicloud.com/intl/en-us/product/modelarts.html, 2022.
[83] Google, "Ai and machine learning products || google cloud," https://cloud.google.com/products/ai, 2022.
[84] Intel, "Confidential VMs on Intel CPUs: Your new intelligent defense ," 2023.
[85] PCI-SIG, "PCI Express 6.0 Specification."
[86] ——, "Integrity and Data Encryption (IDE) ECN Deep Dive," accessed 2023-05-04.
[87] Intel, "Intel tdx connect architecture specification," https://www.intel.com/content/www/us/en/content-details/773614/intel-tdx-connect-architecture-specification.html, 2023.
[88] A. Holdings, "Introducing arm confidential compute architecture guide version 3.0," https://developer.arm.com/documentation/den0125/0300/?lang=en, 2023.
[89] sifive, "RISC-V Security Architecture Introduction," 2019.
[90] C. Consortium, "Compute Express Link® (CXL®): Link-level Integrity and Data Encryption (CXL IDE) ."