SoK: Limitations of Confidential Computing via TEEs for High-Performance Compute Systems

Ayaz Akram
UC Davis
yazakram@ucdavis.edu

Venkatesh Akella
UC Davis
akella@ucdavis.edu

Sean Peisert
UC Davis & LBNL
sppeisert@lbl.gov

Jason Lowe-Power
UC Davis
jlowepower@ucdavis.edu

***Abstract*—Trusted execution environments (TEEs) are primary enablers of confidential computing. This paper presents a systematization of the existing trusted execution environments in industry and academia. We highlight the common mechanisms these TEEs employ to provide different security guarantees and offer a detailed comparative analysis of different TEE proposals. TEEs are anticipated to be a promising solution for the security challenges in the high-performance computing (HPC) domain. However, this paper shows why the existing TEEs are unsuitable for high-performance computing systems. Finally, we present our call for action to work to evolve the TEE technologies with the evolving high-performance computing landscape.**

***Index Terms*—trusted execution environment (TEE), high-performance computing (HPC), confidential computing.**

I. Introduction

Confidential computing (which refers to the use of hardware-enforced (cryptographic) protection of data in **use** in contrast to the data at **rest** (storage) or in **transit** (I/O)) has recently emerged as a new paradigm of computing [1], [2]. Confidential computing creates trustworthy systems rather than point-wise solutions against particular attacks. There are two primary ways to enable confidential computing: privacy-preserving computation techniques (like homomorphic encryption[1] and multi-party computation[2]) and trusted execution environments (TEEs). A comparative analysis of these techniques suggests that hardware TEEs generally incur much lower performance costs than other methods like homomorphic encryption and multi-party computation [6]. TEEs scale well to larger data sizes [2] and generally provide several additional security properties like attestability and code confidentiality (as shown in Table I). Trusted execution environment is defined as follows by the Confidential Computing Consortium [2]:

> "A Trusted Execution Environment (TEE) is commonly defined as an environment that provides a level of assurance of data integrity, data confidentiality, and code integrity. A hardware-based TEE uses hardware-backed techniques to provide increased security guarantees for the execution of code and protection of data within that environment."

Table I provides a set of common properties which are held by a TEE. Figure 1 shows how a trusted execution environment will create a zone of trust for the sensitive application and its data on a general-purpose computing node (Section IV will discuss different mechanisms that are used today to enable this zone of trust).

[1]*Homomorphic Encryption:* A form of encryption that allows computation on ciphertexts, generating an encrypted result which, when decrypted, matches the result of the operations as if they had been performed on the plaintext [3], [4].

[2]*Multi-party Computation:* A subfield of cryptography with the goal of creating methods for parties to jointly compute a function over their inputs while keeping those inputs private [3], [5].

TABLE I: Primary security properties of TEEs

Property	Definition
Data Confidentiality	unauthorized view of data is not allowed.
Data Integrity	unauthorized entities are not allowed to alter the data.
Code Integrity	unauthorized entities cannot alter code in the TEE.
Code Confidentiality	unauthorized entities are not allowed to view the code inside the TEE.
Authenticated Launch	enforcement of authorization checks before process launch.
Programmability	if this is a TEE with arbitrary code or fixed function (code).
Attestability	if a TEE can provide evidence/measurement of its origin & current state.
Recoverability	if a TEE can be recovered from a compromised state.

A. Security in High Performance Compute Environments

Generally, computational scientific research is carried out in high-performance computing (HPC) centers, which provide the required computing and storage resources to users (researchers). The scientific computing problems are large-scale computational problems, which involve large data sets as well. This data is often provided by a third-party (*data provider*) and can be sensitive (fully or partially). Table II provides a few examples of scenarios where different data providers might provide some sensitive data to researchers to perform some type of analysis through their applications.

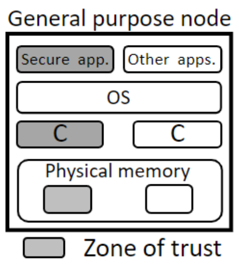

Fig. 1: Trusted execution in traditional computing systems.

TABLE II: HPC Use Cases

Domain	Data Provider	Data types	Applications
Health care	Hospital	Health records, medical images, gene sequences	Machine learning models
Transportation	Public transportation authority	Driving routes	Graph analysis
Energy	Utility company	Home & building energy usage	Real time demand/response

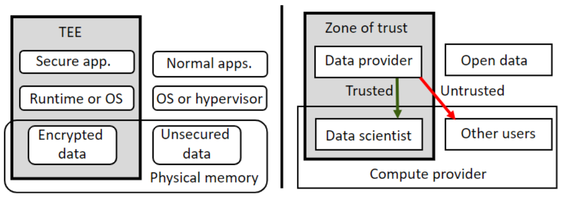

Fig. 2: Creating a zone of trust for sensitive data in HPC centers. Left half shows a general TEE and the right half shows how that TEE can be used to enable a data scientist to compute on sensitive data provided by a trusted data provider and keep it secure from other entities in the system.

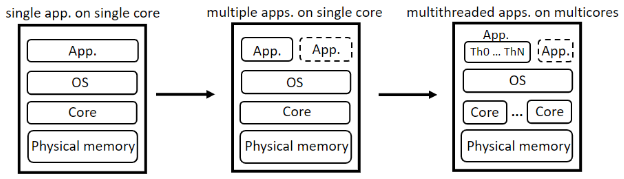

Fig. 3: History of the computing landscape. This figure shows the evolution process of traditional high-performance computing systems. Computing systems have evolved from single processes on a single-core system to multi-threaded applications on heterogeneous multi-core systems.

The use of sensitive data in HPC centers makes it imperative to build HPC systems which can be secure and can be trusted by all the entities involved in a successful scientific workflow. These entities include, multiple users (who might be sharing all or a part of HPC system resources), HPC platform provider, and the data provider. HPC platform providers have tried to tackle this problem already. However, the current solutions can have significant usability challenges as the sensitive data might be hosted in specialized computing facilities and might require specialized access protocols which can be cumbersome for the users/researchers. In contrast, TEE based security solutions can be built in normal computing facilities (without any restrictions on users' access mechanisms). Figure 2 provides a high-level picture of how TEEs can help to create a zone of trust for sensitive data in HPC centers.

B. HPC vs. Cloud Systems

Our focus of discussion in this paper is high-performance computing systems (like those that might find usage in HPC centers e.g., DOE national labs). In contrast to traditional virtualized cloud systems, these HPC systems primarily focus on performance over manageability. Multiple (sometimes heterogeneous) nodes, many cores per node, and integrated accelerators are some characteristics of the HPC systems. Moreover, the applications running on these systems often bypass the OS for performance reasons. In contrast, cloud systems usually rely on adding new privilege layers.

In this paper, using the existing literature, we identify the common mechanisms trusted execution environments (TEEs) use to isolate a sensitive application and its state from the rest of the system. We show how the existing mechanisms do not fit well with the modern **high-performance computing** systems and what are the most promising directions to pursue to ensure that the high-performance computing systems can maintain isolation of sensitive applications [7], [8]. This paper will help the community understand the critical challenges the confidential computing paradigm faces in the context of HPC systems.

C. Contributions

Our main contributions in this work are:

- We provide a categorization and systematization of existing trusted execution environments (TEEs).
- We group TEEs based on the key mechanisms/ideas they rely on to figure out the underlying principles that confidential computing is based on today.
- We take a glance at the traditional computing systems and how they evolved. Using our observations, we point out many ways in which existing TEE technologies would not fit with modern high-performance computing systems: 1) large application modifications, 2) large TCB, 3) focus on core-level execution, and 4) inconsideration of side-channels.
- We provide a call for action on future research that can enable TEEs to be used for high-performance computing systems.

II. Computing Landscape

A. History

Most of the protection mechanisms were invented when the computing system model was very different from today. Therefore, it is essential to look at the history of the computing landscape and its evolution over time. Initially, the computing system model was a single machine with a single core running one application (shown in the left-most part of Figure 3). The OS would make the application feel that it is the only thing running on the system. This model had a vast TCB (you had to trust all the components in the system). Over time, multiple applications started to share the hardware (still a single-core system, as shown in the middle part of Figure 3). The OS time multiplexed the applications on the same hardware. The OS started implementing virtual memory (VM) abstraction to isolate one application from the others. Eventually, the computing systems evolved such that the processor became a multi-core processor (shown in the rightmost part of Figure 3). Applications evolved and started to have more than one execution thread. In this model, the OS would manage multiple threads of execution on multiple cores. In summary, the systems became much more complex to manage; however, they still used the VM-based isolation primitives to ensure isolation among applications on these multi-core systems.

Traditionally, operating systems were responsible for managing most aspects of memory, IO, and computing. The virtual memory (VM) subsystem used for isolation also provided applications an abstract view of physical memory, allowing them to under-subscribe or over-subscribe physical memory. The VM subsystem evolved, but the **coupling of isolation provision, and resource management** stayed intact.

B. Current Computing Landscape

Next, we discuss the current (and future of) computing landscape. In Section V, we will discuss how these advancements in the high-performance computing landscape become the reason for limited applicability of current TEE technologies for HPC. Figure 4 shows a system level view of modern high performance computing systems.

Accelerator Integration: Computing systems have started to integrate different types of accelerators with general-purpose CPUs. In modern computing systems, devices like GPUs, FPGAs, and other accelerators have become a part of the VM subsystem and share virtual address space with applications running on the CPU. The memory allocation is still managed by the OS on the CPU, as the accelerators do not run an OS.

Heterogeneous Memory Systems: Heterogeneous memory systems have become much more prevalent today and rely on emerging memory technologies and more traditional DDR memories. For example, Intel's Sapphire Rapids [9] will include an HBM, a DDR5, and a (byte addressable) NVM (non-volatile memory). The rationale behind using different memory technologies is to allow for different memory types to be used for different applications or phases of a single application. This trend of heterogeneous memory systems makes it necessary to have the ability to migrate data from one device (physical address) to the other (physical address).

Remote Memory Systems: With an increasing adaption of systems where the memory might live remotely (be the NUMA systems or disaggregated systems), memory management might not be entirely done by a (local) OS. Memory management might rely on some remote software/hardware. Network interfaces today have started to rely on RDMA (remote DMA), which bypasses OS mostly and copies the data into a process's virtual address space directly.

Highly Multi-threaded Applications: Especially, in high performance computing systems, the applications are composed of multiple threads, which might execute on multiple cores. Traditionally, the threads used to execute on homogeneous CPU cores, and OS used to manage what threads would execute and where would they execute. In the modern computing systems, the use of accelerators, scale out architectures, and disaggregation of memory resources lead to new models of computing. The host OS might not control all the threads of execution for an application.

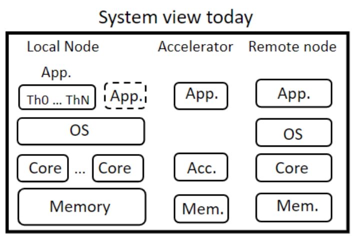

Fig. 4: Modern high performance computing systems. Applications on these systems scale across local node, (integrated or remote) accelerators, and remote nodes.

Direct Memory Access by Devices: Historically, whenever the devices needed to access application's memory, they had to do that via OS as well. All the memory accesses from/to the device have historically being intercepted by the OS. This is not true anymore for high performance computing systems. DMA and RDMA allow direct access to memory by the devices.

III. HPC Focused Trusted Execution Environments

Considering the evolving computing landscape and the security concerns for high-performance computing environments we point out important requirements that the secure architectures focused on HPC should fulfill. We focus on these requirements looking from the outside, i.e., these requirements need to be met to make TEEs usable in HPC set-ups. Following are these requirements:

- R1 *Requirement 1:* HPC-focused TEEs should have minimum performance impact on HPC-style workloads (heavily multi-threaded and have large working sets).
- R2 *Requirement 2:* TEEs should not require application modifications or linking against special libraries as HPC applications often rely on third party libraries.
- R3 *Requirement 3*: HPC-focused TEEs should exclude most of the OS from the TCB. Since, modern HPC applications often bypass the OS for performance benefits (by handing I/O in user-space libraries), reliance on OS security primitives should be minimal.
- R4 *Requirement 4:* HPC-focused TEEs should be capable of expanding across compute nodes as HPC applications mostly scale across multiple nodes and rely on message passing run-times like MPI for communication across these nodes. Moreover, HPC centers (like data centers) are expected to rely on disaggregated architectures (e.g. pooling of memory resources), to increase the utilization of compute/memory resources and save the cost. An HPC focused TEE should consider disaggregated resources in its threat model as well.
- R5 *Requirement 5:* HPC-focused TEEs should enable enclaves which can scale to processing elements other than the general purpose CPUs.

There exist multiple TEE technologies today. In the next section, we will analyze if these existing techniques fulfill the previously mentioned requirements of an HPC-focused TEE.

IV. Systematization of Trusted Execution Environments (TEEs)

In this section, we systematize and classify the existing TEEs into different categories.

Generally, TEEs provide complete control over the trusted computing base (TCB) [1]. The data/code confidentiality and integrity properties of a TEE are usually enabled by isolating an enclave's memory (via the zone of trust shown in Figure 1) from the rest of the system while an enclave is in use. Before providing a classification of TEEs, we will first look at some of the common primitives TEEs use to isolate an enclave from the rest of the system. We also discuss the mechanisms/ways the software or other hardware components use these primitives.

1) Page Table Entry Metadata: Page table entry-level metadata refers to any physical page metadata that TEEs might maintain to identify an enclave page in the hardware. Multiple TEEs rely on this information to implement access control mechanisms. For example, SGX [10] maintains EPCM (enclave page cache map) entries which keep track of the enclaves that own the pages in EPC (enclave page cache), along-with information on the validity of the EPC page. Only SGX instructions can update the entries in the EPCM; therefore, the system software can track any unwanted change in the enclave's address map. Another example is AMD SEV [11], which uses bit 47 of the physical address in a page table entry to identify whether this page is secure. The hypervisor or the host OS manages this bit.

2) Encryption: Encrypting physical memory is a very common primitive used by TEEs (e.g., AEGIS [12], SGX [10], Graviton [13], HETEE [14], ARM RME [15]) to ensure confidentiality of data belonging to an enclave (and can be a strong mitigation against physical attacks). Usually, the TEEs rely on an encryption key which is generated and stored in an (isolated) trusted processor (or some hardware component). For example, AMD SEV [11] relies on an ARM-based processor (AMD Platform Security Processor) for key management. As the cache blocks move from/to the processor chip to/from the DRAM, the key is used to encrypt/decrypt the cache blocks transparently.

3) Physical memory isolation via ISA extensions: RISC-V based TEEs (e.g., Keystone [16], CURE [17], Elasticlave [18], TIMBER-V [19]) mostly rely on PMP (physical memory protection) ISA extension. PMP controls U (user) and S (supervisor) modes' access to certain memory regions. The allowed access (r-w-x) permissions and the memory region can be configured using PMP address (pmpaddr) and configuration (pmpcfg) registers. There also exist proposals for providing physical memory protection to IO devices via IOPMP [20].

4) Use of tags/identifier in hardware: Some TEEs also use a tag or identifier to distinguish enclave data from other software in the system (for access control). For example, CURE [17] uses enclave ID for bus arbitration to enable enclave to the peripheral binding. For this purpose, CURE [17] hardware relies on a filter engine on the system bus. Bastion [21] depends on a *module ID*, which is a new component in caches and TLB and acts as a tag for the currently executing process. ARM TrustZone [22] uses a single-bit identifier to distinguish between the secure and non-secure world (for device communication). SiFive's WorldGuard [23] can tag bus transactions to differentiate between software contexts that originated a request, allowing the target to determine if it trusts the requestor. HECTOR-V [24] also relies on identifiers embedded in interconnects, which helps create a safe IO path. AMD SEV [11] hardware tags all code and data with its VM ASID (inside the SoC), indicating the VM, which is the data owner.

5) Privileged Software/Hardware: Trusted execution environments mostly do not trust the OS and try to bypass the OS privileges. They usually do this through additional hardware/software components to perform privileged operations focused purely on security. For example, CURE [17] uses a hardware-based security monitor to monitor the system bus's access. Keystone [16] uses an M-mode software-based security monitor to manage physical memory isolation primitive (i.e., PMP). Similarly, ARM RME (realm management extension) [15] relies on a monitor to enforce its security guarantees.

A. Classification of TEEs

We present a classification taxonomy of existing trusted execution environments to enable a better understanding of the vast design space that is covered by TEEs. Figure 5 show this taxonomy with some examples of TEEs from each class. TEEs can be classified based on different factors. We use the following factors for this classification:

- *Isolation level:* This defines at what level the secure and non-secure components are isolated.
- *Threat model configurability:* This determines if the threat model of a TEE can be configured (either at the run time or the implementation time).
- *Enclave privilege level:* This is the privilege level at which the enclave operates.
- *Openness:* This determines if the TEE is an industrial solution or academic. Usually the industrial solutions are closed-source and academic solutions are open-source.
- *Abstraction level:* This is the level of abstraction at which the TEE provides an interface to the user.
- *Target computing:* This is the type of computing which the TEE is mainly designed for.

A more systematic and detailed comparison of different TEEs is provided in Table V (in Appendix section). Here, we provide some discussion and observations on different classes of TEEs that are shown in Figure 5 with important examples.

We observe that the current TEEs which provide the highest isolation level usually achieve it via physical isolation or partitioning at a very coarse granularity. For example, AWS Nitro enclaves are an example of highly isolated enclaves which provide (constrained) enclave virtual machines with no storage, network, or interactive access [25]. AWS Nitro enclave has only a single point of connection to the outside world which happens via bi-directional VM socket (*vsock*) between the parent instance and an enclave [25]. The major drawback of highly isolated enclaves is the difficulty to use them. For example,

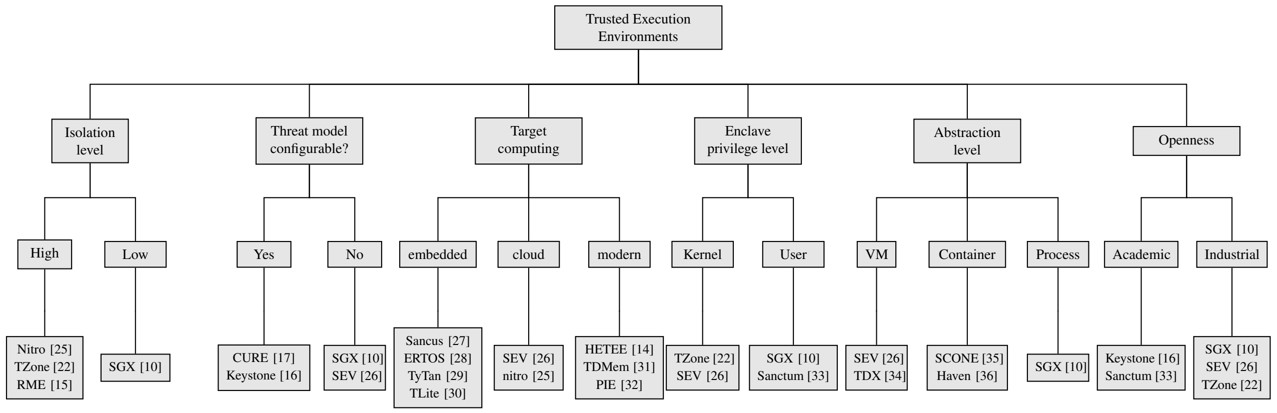

Fig. 5: Classification of TEEs and references to some examples of each class

applications will have to rely on message passing, RPC or micro-services to interact with their secure compartment on the enclave virtual machine (in case of AWS Nitro). Other examples of highly isolated enclaves, ARM TrustZone [22] and Realms [15], partition the entire physical address space at a very coarse granularity (into secure and non-secure worlds).

There are also many examples of configurable TEEs (which can lead to variable TCB sizes). Configurability is a desirable property in the current heterogeneous world. Applications executing on a modern (heterogeneous) HPC system might not have the same sensitivity level (or require the same security guarantees, e.g., integrity is not essential if the application is not going to reuse previously written data). CURE [17] provides the ability to define enclave trust boundaries (at different granularity levels). AEGIS [12] provides the ability to have both a trusted and untrusted OS. ShEF [37], a trusted execution environment for FPGAs, provides the ability to customize encryption logic parallelism and authentication block size.

TEEs could opt for a specific security vs. cost tradeoff depending on the computing type they are targeting. We observe that most of the earlier TEEs focused on general-purpose desktop/cloud or embedded computing (e.g., [11], [22], [27]). However, some recent examples of academic proposals target parts of modern computing systems. For example, HETEE [14] targets server rack-scale computing. Graviton [13] and HIX [38] tried to enable isolated execution on GPUs, ShEF [37] targets FPGAs and TDMem [31] focused on RDMA-based disaggregated systems.

Privilege-level based classification divides enclaves into kernel-space or user-space enclaves. Kernel-space enclaves can run trusted kernel-mode software inside the enclave, which means that these enclaves generally have a large TCB. For example, Keystone [16] requires having a kernel-space runtime (for user-space application's resource management) inside the enclave. SEV [11] allows kernel-space enclaves, where the guest OS is a part of the enclave. SGX [10], on the other hand, is a user-space enclave and has smaller TCB (compared to SEV). However, user-space enclaves have to pass the (trusted) user-space and (untrusted) kernel-space boundary to perform system-level services, which can also have security concerns.

V. Limitations of Existing TEEs

Next, we discuss some limitations of existing trusted execution environments and show how they hinder the adoption of secure execution environments for modern computing systems.

Confidential computing environments rely on hardware primitives to protect/isolate an enclave's memory from the rest of the system. These primitives can sometimes impose restrictions on the system's resource management, decreasing the usability and efficiency of the system. We present the following observations on the kind of limitations confidential computing can impose on modern HPC environments:

A. *Heavy Application Code Modifications*

Today, we do not have primitives that allow fine separation of management and protection. As a result, the entire OS is mostly not trusted in the confidential computing threat model.

Therefore, the programming model of most of the TEEs provides limited support for traditional C libraries (e.g., simpler *muslc* replaces more complicated *glibc*). The programming model requires heavy modifications in userspace applications (especially the large applications which run on high-performance computing systems) and limits their functionality.

B. *Large Trusted Compute Base (TCB)*

Since the OS is mostly not trusted in the confidential computing threat model, delegating resource management to the OS can lead to vulnerabilities. For example, managing an enclave's address space allows a malicious OS to launch page fault-based attacks on enclaves. These attacks are possible because OS can modify access permission of enclave's pages, which would lead to page faults, and thus OS can figure out the enclave access pattern. Such attacks, called *controlled channel attacks* [39] are deterministic (and noise-free) and can have large leakage bandwidth compared to other noisy side channels. The proposed solutions to the controlled channel attacks require the enclave to control its page tables and enforce secure-paging policies within an enclave. Examples of such proposals include Autarky [40], Keystone [16], and CURE [17]. The drawback of these approaches is that they lead to a *larger TCB* and more complexity in the enclave.

The scheduling and synchronization of threads by an untrusted OS can lead to multiple security issues[3]. For example, an untrusted OS can influence a machine learning model leading to poisoning attacks by controlling the order in which the threads of the training algorithm are executed [41]. To solve this problem, some TEEs have implemented limited thread handling inside the enclave, which might reduce the system's efficiency overall. For example, enclaves (like SGX [10]) might enforce a static number of threads because they might only allow statically-defined entry points for executing threads. Many of the TEEs based on SGX have similar limitations. Enclaves like Keystone [16] do not support multi-threaded execution at all at the time of writing this paper. In summary, today's enclaves generally do not have good support for multi-threaded execution unless they are willing to have a *large TCB*. Interestingly, virtual machine (VM) based enclaves include a guest OS in the TCB and allow multi-threaded applications to run transparently. Not only do the VM-based enclaves have a very large TCB, but multi-threaded execution in virtual machines can also have significant performance implications. For example, when threads yield during synchronization operations, they can cause costly KVM exits [42], [43].

C. Focus on Core Level Execution

A significant limitation of most of today's TEEs is that they focus on a *core-level* view of memory permissions. This behavior limits the applicability of the TEEs to heterogeneous HPC systems.

The lack of an OS or another privileged software on accelerators makes it very hard to take the memory management controls from the host OS. In addition, accelerators are generally more sensitive to address translation latencies [44], so access control through additional privileged components is complex. Therefore, if the memory level view of access control is unavailable, it is hard for different compute elements to share the same trusted memory. This behavior is also true for disaggregated/remote memory systems. In those systems, the host OS and other privileged software or component on the host node would not entirely control the remote parts. It becomes hard to rely on a core-level execution view to ensure security.

Another implication of *core-level* view of memory permissions is that they require synchronization of memory permissions across the cores, which are used to execute all the threads of an application. This synchronization is costly [45], and today we do not have good hardware primitives (currently, inter-processor interrupts can be used). Moreover, the synchronization becomes even more challenging when the application scales to accelerators or remote compute nodes.

Devices also suffer because of the core-centric behavior of current TEEs. Today, most TEEs use untrusted (shared) memory buffers (e.g., bounce buffers in Linux) as temporary storage for the data moving between the devices and an enclave. The use of temporary buffers leads to extra copies of the data and has performance implications as well. This behavior also implies that the DMA functionality would not work securely with current TEEs.

[3]scheduling based denial of service attacks are common, but generally not a part of the threat model of confidential computing systems.

D. No Consideration of Side Channels

TEEs do not consider system components that are not memory or cores. In other words, current TEEs generally do not focus on things that are not architecturally visible. This behavior allows cache or system-bus-based side channel attacks possible on TEEs [46]–[48]. High bandwidth leakage channels can be possible, especially on modern high-performance computing systems with high-bandwidth links between physically isolated components.

E. Other Limitations

This subsection briefly discusses other (less critical) limitations of TEEs that do not fit in any of the above categories.

1) Memory Isolation Primitives and Fragmentation: Most of the hardware primitives used by TEEs today limit the maximum number of enclaves possible or cause fragmentation issues. For example, ARM TrustZone [22] uses an address space controller to create an OS hypervisor mapping. Keystone [16] relies on contiguous physical memory for an enclave (as PMP defined physical memory range has to be contiguous). Similarly, CURE requires the physical memory of an enclave to be contiguous. If multiple enclaves are executing simultaneously, the requirement of contiguous physical memory for each enclave can cause fragmentation and potentially overuse of resources.

2) Limitations on maximum number of enclaves: The memory isolation primitives used by the existing TEEs can also limit the maximum number of executing enclaves simultaneously. For example, PMP-based TEEs (like Keystone) cannot have more enclaves than the number of PMP entries (latest specifications allow upto 64 entries [49]). Similarly, AMD SEV has limitations on the number of maximum enclave VMs. The maximum number possible on the AMD EPYC system was 15 due to a fixed number of slots for encryption keys (one needed for each enclave VM) in the memory controller [50]. Sanctuary [51] also has a limitation on the maximum number of enclaves due to address space controller constraint [52]. CURE [17] can support 13 enclaves concurrently due to limitations of the hardware arbiter they use. The limitation on a maximum number of enclaves can be a big issue for multi-tenant computing systems.

3) Limitations on data movement: Cryptographic isolation primitives inhibit the transparent data movement in heterogeneous memory systems. For example, to ensure that two same plain text pages at different physical addresses have different cipher texts (as a protection mechanism against cipher-text block move attacks), AMD SEV uses a physical address-based tweak algorithm [53], [54] which uses a block's physical address and an encryption key (xor-encrypt-xor tweak [55]). Since the host-physical address is used to determine the cipher-text of a page, the hypervisor cannot move a page between the two physical addresses once it is allocated to the secure VM. The hypervisor has to lock the physical pages in memory which leads to pre-allocation

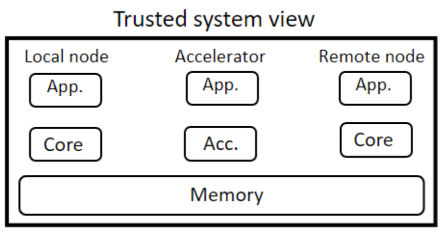

Fig. 6: Required trusted system view

of all the required physical memory and can cause under-utilization of resources and unintended effects on NUMA affine workloads [42]). The transparent movement of physical pages from one device to the other would have required the data to transit via the memory controller so that it can be decrypted and re-encrypted again using the new physical address, which can be costly.

4) Compute on Modern Computing Systems: The state of an enclave or secure process on context switch cannot be protected easily if the enclave is scaling across multiple computing elements (some of which might not even have an OS), which can be physically distant from one another.

VI. Promising Future Research Directions

In this section, we discuss the promising research directions enabling confidential computing on high-performance computing systems. These research directions can help in mitigating the critical limitations of today's TEEs: heavy application changes, large TCB, core-level isolation view, and inability to protect against side-channels.

Figure 6 shows the trust model we need for modern high-performance computing systems (like the one shown in Figure 4). The local (general purpose) node, accelerator node, and remote node share part of the trusted memory and should not need to trust any component other than the core/s they are executing on.

We argue that given the way computing is evolving, we do not necessarily treat security and performance as a trade-off, but we can achieve both together. We emphasize that the system view in Figure 6 also fits well with the optimizations which can extract more performance from a computing system. Therefore, synergistically building secure and performant systems is the right thing. A similar observation by Orenbach et al. [56] suggests that enclaves have many similarities with accelerators: significant invocation overheads, space constrained private memory, and inability to directly invoke OS services such as network and I/O. The solutions to these problems for accelerators [57], [58] mostly involve bypassing the OS (for performance reasons), which is an attractive property for enclaves as well.

We discuss some of the promising future directions:

A. New hardware primitives

Most of today's commonly used operating systems (Linux, Windows, macOS) use a monolithic kernel, which provides a fixed abstraction to the user-mode applications and depends on a generic or a homogeneous view of applications while managing their resources. However, many other kernel designs have been proposed in the past. For example, exokernel [59] proposed an idea of application-level resource management. The applications again are pursuing this direction by trying to relinquish the monolithic nature of the kernel. Today, the applications are mainly doing this for performance reasons.

For example, in modern computing systems, HPC applications often bypass the OS and handle I/O in user-space libraries or run-times via a specialized HPC networking stack (e.g., RDMA [60] via InfiniBand [61]). The noteworthy point is that these libraries tend to make certain assumptions about HPC. For example, modern MPI libraries assume that they can fully utilize CPU cores to spin on network hardware resources to check for progress. Less dependence/reliance on a (primarily untrusted) OS fits well with the confidential computing model. Moving the resource management code to user space also means a larger software TCB for the enclave.

The increasing code bloat implies that the software will always have exploitable bugs. On the other hand, hardware has more trust due to two primary properties: immutability and privilege [62]. Therefore, we emphasize focusing on increasing the hardware TCB components. We believe that there is an opportunity for computer architects to think of new hardware primitives as the ones we have today do not work well, as discussed in Section V.

B. Horizontal privilege levels

As discussed before, the vertically integrated model of privilege levels does not work for evolving high-performance computer systems. We emphasize horizontal privilege levels, where there can be a variable number of vertical layers in each horizontal privilege level (e.g., there might not be an OS in the privilege hierarchy of an accelerator). Depending on the threat model, one horizontal layer can have more privilege than the other. An example of a similar system is ARM TrustZone [22] which divides the system into a secure and a normal world. However, it does that only for a CPU system and cannot create a secure world for accelerator or other remote computing elements/memory nodes.

C. Data centric enclaves

Current TEEs, when trying to isolate software from the rest of the system (via TEE-based containers, for example), end up with usability constraints and eventually try to emulate existing system components (like POSIX or devices) inside the contained systems and eventually have to deal with the same problems they started with [63]. Since today's threat models mainly consider untrusted software, the "unit of protection" should be individual data items [63]. Data-centric enclaves can solve this problem, which inherently rely on memory/data level isolation view rather than core-level isolation view. One example of such architecture is *Border Control* [64] which keeps the protection checks of IO-MMU consistent with the TLB checks via a hardware structure. This way, *Border Control* can maintain the memory level view of permissions and protect against accelerator-based attacks.

D. Capability based enclaves

The idea of capabilities (first proposed a few decades ago [65]) provides ways to enable compartments at different

TABLE III: Survey of Attacks on TEEs/Enclaves

Type of attacks	Examples
Side channel	[46]–[48]
Controlled channel	[39], [69]
Encryption Attacks	[53], [54]
IO Based Attacks	[55], [70]

TABLE IV: Example of Tools/Frameworks for TEEs

Type of tool	Examples
TEE Containers	Graphene [71], SCONE [35], SGX-LKL [72]
Simulation	FireSim [73], gem5 [74]
Profiling	TS-Perf [75], sgx-perf [76] , Tee-perf [77]

abstraction layers of computing systems. Capability based machines have also existed for a long time now (e.g., M-Machine [66], Rice research computer [67], CHERI [68]). Capability based architectures inherently do not have to follow a vertically integrated privilege hierarchy, rather these architectures rely on capabilities of different components in the system to perform access control checks. Therefore, using capabilities to implement enclaves seems a promising idea for (heterogeneous) high-performance computing systems.

VII. A Brief Discussion on Topics Beyond the Scope of this SoK Study

In this section, we briefly discuss few topics that are not in the main scope of this paper.

A. Survey of Attacks on TEEs/Enclaves

There is a lot of research on bypassing the security guarantees of TEEs/enclaves. Table III shows some of the attacks that are possible on TEEs/enclaves.

1) Protection Against Side Channel Attacks: TEEs primarily do not consider side channels. However, there are a few exceptions [17], [33]. Komodo [78] obliviates computing to protect against side channels. Sanctum [33] protects against cache side channels by enforcing distinct cache sets per enclave. Keystone [16] also allows the ability to include side channels in the threat model.

2) Protection Against IO Attacks: Since IO devices are generally not a part of the CPU package and are not trusted, they can come from a malicious vendor. Such devices can break the confidentiality of the enclave's data when it leaves the CPU package. For example, Lee et al. [70] presented an off-chip attack on enclaves by snooping the memory bus. Some of the TEEs (e.g., CURE [17], HECTOR-V [24]) use enclave to peripheral binding to protect against IO-based attacks.

B. Tools for TEE Platforms

Table IV provides a brief survey of different kinds of tools/infrastructure that can help the usage of TEEs or advance research on TEEs. TEE containers help in the execution of unmodified code on TEEs. These containers often provide an emulated view of specific system components and might have many of the limitations of the underlying TEE. Application profiling helps to understand their behavior better and potentially optimize their execution on given hardware. Standard profiling tools might not be able to interact with the enclave applications due to the specialized execution mode of enclaves. Though some specialized profiling tools exist for enclaves, as shown in Table IV, there is still room for improvement in this space. Simulation support for TEEs is essential. TEEs usually rely on a hardware-software co-design. However, most of the architectural simulators are not full-system and might not be able to support all components needed to simulate a TEE. On top of that, the details of the targeted TEE might not be openly available. However, there are options for the simulation of RISC-V-based TEEs (as shown in Table IV).

C. Formal Verification of TEEs

Formal verification provides means to evaluate if a security mechanism is correct and does what it claims. There are a few examples of TEEs which have been formally verified. Komodo [78] and Sanctum [33] are a couple of examples of TEEs with a formal proof of their correctness. RISC-V's PMP (which is used by many TEEs, e.g., Keystone [16]) has also been formally verified [79].

VIII. Conclusion

In this paper, we provided a systematization study of existing trusted execution environments (TEEs) which are one of the main enablers of confidential computing. We discussed the primary mechanisms or primitives the existing TEEs use. We also provided a list of the limitations of the existing TEEs, which we believe are the main reasons why the current TEEs are not suitable for high-performance computing. The existing primitives to build TEEs require large application modifications, lead to large TCB, focus on core-level execution, and do not consider side channels a part of their threat model. These limitations make it very hard to run HPC applications under TEEs, cause significant slowdowns for HPC workloads, and do not ensure their security due to an insufficient threat model. We believe the existing TEE technologies are point solutions for different computing targets. And in the future, we need to either generalize the TEE technologies to be able to use them for any computing domain or come up with point solutions focused on high-performance computing. We also provided a list of the directions we believe can enable TEEs to be a good fit for high-performance computing systems. We believe that this document can provide the community with essential insights to guide their future research on confidential computing technologies in the right direction.

APPENDIX: SUMMARY OF COMPARATIVE ANALYSIS OF TEEs

Table V. Taxonomy of different TEE features. Each column shows the status of a particular TEE property and the mechanism to achieve that property is shown in parentheses. Blank entries indicate that the information on that property was not available.

TEE	Data Conf.	Data Integrity	Code Integrity	Code Conf.	Auth. Launch	Attestability	Custom.	Isolation	Software Attacks[1] Protected	Hardware Attacks[2] Protected	Side Channel Protection	TCB	Level[3]	Changes Needed (HW,SW)	IO Handling[4]	Use Cases	Notes
Autarky [40]	✓	✓	✓	✗	✗	✓	✗	✗	✗	physical attacks	controlled channels	CPU, OS Driver, LibOS	process	(✓, ✓)	clear	desktop/cloud	based on SGX
AWS Nitro [25]	✓(P, VM)	✓(P, VM)	✓(P, VM)	✓(P, VM)	✓(E)	✓(E)	✗	✓(VM)	other VMs	✗	✗	VM, OS, Nitro HV	VM	✗, ✓	vsock	cloud VMs	highly isolated
AEGIS [12]	✓(PTR, E)	✓(IT)		✓(PTR)	✓(H)	✓(H)	✓	✓	processes, OS	physical attacks	✗	CPU, OS	processes	(✓, ✓)	clear	desktop/cloud	academic
Bastion [21]	✓(MP)		✓	✓	✓	✓(H)	✗	✓(IA, MP)	processes, OS	physical attacks	✗	CPU, HV, gOS	VM	✓, ✓		cloud	dynamic RoT, UltraSPARC
CURE [17]	✓(IA)	✓(IA)	✓(IA)	✓(IA)		✓	✓	system bus arbiter (IA)	processes, OS	IO attacks, physical attacks	✓(cache & controlled)	configurable	process (user/kernel space)	(✓, ✓)	✓(enclave to peripheral binding)	variable	target configurability
Elasticlave [18]	✓(MP)	✓		✓		✓	✓	✓(MP)	processes, OS	physical attacks		CPU, SM, RT	process	✗, ✓	secure	variable	RISC-V based
ERTOS [28]	✓(MP)	✓(MP)	✓(MP)	✓(MP)			✗	✓(MP)				ERTOS module of FreeRTOS	tasks	(✗, ✓)		embedded systems	relies on PMP
Graviton [13]	✓(E, I)	✓(MAC)	✓(MAC)	✓(E)	✓	✓	✗	✓(P, IA)	OS, HV, processes		✗	GPU, on-package memory	GPU kernels	✓, ✓		GPU computing	
HECTORV [24]	✓(MP)	✓			✓	✗	✗	✓(SP, IA, MP)	processes	other peripherals on SoC	✗	HW	process	✓, ✗	peripheral binding	heterogeneous systems	RISC-V based systems
HETEE [80]	✓(E)	✓(E)	✓(E)	✓(E)	✓	✓	✗	✓	processes, OS		✗	hardware security controller, PCIE fabric	multi-node (computing elements)	✗, ✗		rack-scale computing	relies on a hardware security controller
HIX [38]	✓(E)	✓	✓	✓		✓	✗	✓	OS, processes			GPU	CPU-GPU applications	PCIE root complex & MMU, GPU driver	secure	GPU (heterogeneous computing)	relies on SGX
Iso-X [81]	✓(P)			✗		✓	✗	✓			✗	HW only	process				OpenRISC, dynamic RoT
IceClave [82]	✓(E)	✓(IT)		✓(E)			✗	✓	processes	physical attacks	✗	embedded processor (in SSD controller)	offloaded applications			in-storage computing (flash based SSDs)	extends TrustZone in ARM processor of SSD controller
Komodo [78]			✓	✓		✓	✗	✓	OS, processes	physical attacks	✗						formally verified
KeyStone [16]	✓(MP)	✓(MP)	✓(H)	✓(MP)	✓(E)	✓	✓	✓(MP)	OS, processes		✓(extension)	SM, CPU, runtime	process (U+S mode)	(✗, ✓)	✗	variable	RISC-V based
ARM Realms [15]	✓(P)			✗		✗		✓(cache access checks)			✗		process			embedded, mobile	uses a security monitor
Sanctum [33]	✓				✓	✓		✓	processes, OS	✗	✓		process				RISC-V based
Sancus [27]	✓			✗	✓	✓					✓	HW only				embedded	hardware only TCB
SecureBlue++ [83]	✓			✓		✗		✓			✗	HW only					POWER
ShEF [37]	✓(E, MAC)	✓(E, MAC)	✓(E, MAC)	✓(E, MAC)	✓(E)	✓(E)	✓	✓	CPU OS, processes		✗	FPGA, shell logic	FPGA bitstream	(✗, ✓)		cloud FPGAs	first work on FPGA TEE
SGX [10]	✓	✓	✓	✗	✗	✓	✗	✗	✗	Small desktop Apps.	large[5]	CPU, SGX driver	process	(✓, ✓)	outside enclave, in clear	desktop/cloud	

SEV [11]	✓(E)	✗	✗	✓	✓	✓(hashing VM)	✗	VM level (through page table's C bit)	processes, OS	physical attacks	✗	gOS, CPU	VM	✗, ✗	clear	cloud	
SEV-ES [84]	✓(E)	✗	✗	✓	✓	✓		VM level	processes, OS	physical attacks	✗	gOS, CPU	VM	✗, ✗		cloud	protects VM execution state
SEV-SNP [85]	✓	✓(nested paging)	✓(nested paging)	✓	✓	✓	✗	VM level	processes, OS	physical attacks	✗	gOS, CPU	VM	✗, ✗		cloud	not very strong integrity guarantees
Penglai [86]	✓(MP)	✓(MT)	✓	✓			✗		OS, processes			CPU	process				targets scalability, RISC-V based
TDX [34]	✓	✓	✓	✓				VM level	HV, OS, processes				VM			cloud VMs	
TDMem [31]	✓(MP, E)	✓(MAC)			hashing of FPGA bit-stream		✓(FPGA based)	✓(MP, address translation tables)	memory access attacks from the donor and donee		✗	kernel (donee only), FPGA boards	RDMA disag-gregated systems	needs FPGA, ✓(kernel)		cloud dis-aggregated systems	one of the earliest works on disag-gregated systems
TrustZone [22]	✓(P)	✗	✗	✓(P)		✗	✗	✓(cache access checks)	non-secure world	✗	✗	CPU, secure OS	process	(✓, ✓)	clear	embedded, mobile	AXI bus time slicing
TIMBERV [19]	✓(MP)	✓(MT)	✓	✓		✓(E)	✗	tagged entry points, regions MPU	processes, OS	✗	✗	CPU	process	(✓, ✓)	clear	embedded systems	one of the few to rely on tagged mem.

Note: Conf. : Confidentiality, Auth. : Authenticated, E: Encryption, P: Partitioning, VM: virtual machine, HV : hypervisor, IT: integrity tree, H : hashing, MP : memory protection checks, IA : id assignment
gOS : guest OS, MAC : message authentication code, SP : secure processor, Custom. : Customizability, RoT : root of trust, MT : Memory Tagging
[1]Software attacks have software and [2]hardware attacks have hardware as the attack surface. [3]Level is the granularity/level at which protection is provided.
Other Notes: These TEEs generally do not consider side channels. Threat of side channels depend on the data sensitivity and leakage rate.
SGX provides strong protection against integrity attacks. SEV-SNP provides some gaurantees against inegrity attacks. [4]I/O includes GPUs, accelerators and FPGAs as well.

References

[1] M. Russinovich, M. Costa, C. Fournet, D. Chisnall, A. Delignat-Lavaud, S. Clebsch, K. Vaswani, and V. Bhatia, "Toward confidential cloud computing: Extending hardware-enforced cryptographic protection to data while in use," *Queue*, vol. 19, no. 1, 2021.

[2] "A Technical Analysis of Confidential Computing v1.1," https://confidentialcomputing.io/white-papers-reports/, 2021.

[3] C. C. Consortium, "Confidential Computing Consortium Scope," 2021, retrieved August 5, 2021 from https://github.com/confidential-computing/governance/blob/master/scoping.md.

[4] C. Gentry, "A Fully Homomorphic Encryption Scheme," Ph.D. dissertation, Stanford University, 2009.

[5] A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 1986, pp. 162–167.

[6] C. C. Consortium, "Confidential Computing: Hardware-Based Trusted Execution for Applications and Data," https://confidentialcomputing.io/white-papers-reports/, 2021.

[7] S. Peisert, "Security in high-performance computing environments," *CACM*, vol. 60, no. 9, pp. 72–80, 2017.

[8] ——, "Trustworthy scientific computing," *Communications of the ACM*, vol. 64, no. 5, pp. 18–21, 2021.

[9] N. Nassif, A. O. Munch, C. L. Molnar, G. Pasdast, S. V. Lyer, Z. Yang, O. Mendoza, M. Huddart, S. Venkataraman, S. Kandula *et al.*, "Sapphire rapids: The next-generation intel xeon scalable processor," in *2022 IEEE International Solid-State Circuits Conference (ISSCC)*, vol. 65. IEEE, 2022, pp. 44–46.

[10] V. Costan and S. Devadas, "Intel SGX Explained," Cryptology ePrint Archive, 2016, https://eprint.iacr.org/2016/086.

[11] "Secure Encrypted Virtualization (SEV)," https://github.com/AMDESE/AMDSEV.

[12] G. E. Suh, D. Clarke, B. Gassend, M. Van Dijk, and S. Devadas, "Aegis: Architecture for tamper-evident and tamper-resistant processing," in *ACM International Conference on Supercomputing 25th Anniversary Volume*, 2003, pp. 357–368.

[13] S. Volos, K. Vaswani, and R. Bruno, "Graviton: Trusted execution environments on gpus," in *13th USENIX OSDI*, 2018.

[14] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, B. Zhao, Z. Wang, Y. Zhang, J. Ying, L. Zhang, and D. Meng, "Enabling rack-scale confidential computing using heterogeneous trusted execution environment," in *2020 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2020, pp. 991–1006. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40000.2020.00054

[15] "Realm Management Extension," https://developer.arm.com/documentation/den0126/latest.

[16] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.

[17] 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 *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1073–1090.

[18] 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)*, 2022.

[19] S. Weiser, M. Werner, F. Brasser, M. Malenko, S. Mangard, and A.-R. Sadeghi, "Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v." in *NDSS*, 2019.

[20] P. S.-C. Ku, "IOPMP Updates Protection Of IOPMP Andes Technology," 2021, rISC-V Summit.

[21] D. Champagne and R. B. Lee, "Scalable architectural support for trusted software," in *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*. IEEE, 2010, pp. 1–12.

[22] T. Alves and D. Felton, "TrustZone: Integrated Hardware and Software Security," *Information Quarterly*, pp. 18–24, 2004.

[23] B. Wheeler, "Sifive secures risc-v," *Microprocessor report*, 2019.

[24] P. Nasahl, R. Schilling, M. Werner, and S. Mangard, "Hector-v: A heterogeneous cpu architecture for a secure risc-v execution environment," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 187–199.

[25] "AWS Nitro Enclaves," https://aws.amazon.com/ec2/nitro/nitro-enclaves/.

[26] D. Kaplan, "AMD x86 Memory Encryption Technologies," in *Linux Security Summit*, 2017.

[27] J. Noorman, P. Agten, W. Daniels, R. Strackx, A. Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens, "Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base," in *22nd USENIX Security Symposium (USENIX Security 13)*, 2013, pp. 479–498.

[28] A. Thomas, S. Kaminsky, D. Lee, D. Song, and K. Asanovic, "Ertos: Enclaves in real-time operating systems."

[29] F. Brasser, B. El Mahjoub, A.-R. Sadeghi, C. Wachsmann, and P. Koeberl, "Tytan: Tiny trust anchor for tiny devices," in *DAC*, 2015, pp. 1–6.

[30] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "Trustlite: A security architecture for tiny embedded devices," in *Proceedings of the Ninth European Conference on Computer Systems*, 2014, pp. 1–14.

[31] T. Heo, S. Kang, S. Lee, S. Hwang, and J. Huh, "Hardware-assisted trusted memory disaggregation for secure far memory," *arXiv preprint arXiv:2108.11507*, 2021.

[32] M. Schneider, A. Dhar, I. Puddu, K. Kostiainen, and S. Capkun, "Pie: A platform-wide tee," 2021.

[33] V. Costan, I. Lebedev, and S. Devadas, "Sanctum: Minimal hardware extensions for strong software isolation," in *25th USENIX Security)*, 2016, pp. 857–874.

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

[35] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, D. O'keeffe, M. L. Stillwell *et al.*, "SCONE: Secure Linux Containers with Intel SGX," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*, Savannah, GA, Nov. 2016, pp. 689–703.

[36] A. Baumann, M. Peinado, and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, p. 8, 2015.

[37] M. Zhao, M. Gao, and C. Kozyrakis, "Shef: shielded enclaves for cloud fpgas," in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 1070–1085.

[38] 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.

[39] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, "Preventing page faults from telling your secrets," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016, pp. 317–328.

[40] M. Orenbach, A. Baumann, and M. Silberstein, "Autarky: Closing controlled channels with self-paging enclaves," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020.

[41] J. R. Sanchez Vicarte, B. Schreiber, R. Paccagnella, and C. W. Fletcher, "Game of threads: Enabling asynchronous poisoning attacks," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 35–52.

[42] A. Akram, A. Giannakou, V. Akella, J. Lowe-Power, and S. Peisert, "Performance analysis of scientific computing workloads on general purpose tees," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 1066–1076.

[43] X. Ding, P. B. Gibbons, and M. A. Kozuch, "A Hidden Cost of Virtualization when Scaling Multicore Applications," in *5th USENIX Workshop on Hot Topics in Cloud Computing*, 2013.

[44] B. Pichai, L. Hsu, and A. Bhattacharjee, “Architectural support for address translation on gpus: Designing memory management units for cpu/gpus with unified address spaces,” *ACM SIGARCH Computer Architecture News*, vol. 42, no. 1, pp. 743–758, 2014.

[45] J. C. Mogul, A. Baumann, T. Roscoe, and L. Soares, “Mind the gap: Reconnecting architecture and os research,” in *13th Workshop on Hot Topics in Operating Systems (HotOS XIII)*, 2011.

[46] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida, “Crosstalk: Speculative data leaks across cores are real,” in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1852–1867.

[47] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, “Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution,” in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 142–157.

[48] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, “Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution,” in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[49] “RISC-V Instruction Set Manual,” 2022, https://github.com/riscv/riscv-isa-manual.

[50] *AMD SEV*. [Online]. Available: https://docs.openstack.org/nova/latest/admin/sev.html

[51] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, “Sanctuary: Arming trustzone with user-space enclaves.” in *NDSS*, 2019.

[52] H. Li, W. Huang, M. Ren, H. Lu, Z. Ning, H. Cui, and F. Zhang, “A novel memory management for risc-v enclaves,” 2021.

[53] L. Wilke, J. Wichelmann, M. Morbitzer, and T. Eisenbarth, “SEVurity: No Security Without Integrity - Breaking Integrity-Free Memory Encryption with Minimal Assumptions,” in *2020 IEEE Symposium on Security and Privacy (SP)*, may 2020, pp. 1431–1444.

[54] Z.-H. Du, Z. Ying, Z. Ma, Y. Mai, P. Wang, J. Liu, and J. Fang, “Secure Encrypted Virtualization is Unsecure,” *arXiv preprint arXiv:1712.05090*, 2017.

[55] M. Li, Y. Zhang, Z. Lin, and Y. Solihin, “Exploiting unprotected io operations in amd’s secure encrypted virtualization,” in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1257–1272.

[56] M. Orenbach and M. Silberstein, “Enclaves as accelerators: learning lessons from gpu computing for designing efficient runtimes for enclaves.”

[57] M. Silberstein, B. Ford, I. Keidar, and E. Witchel, “Gpufs: Integrating a file system with gpus,” in *ASPLOS*, 2013.

[58] S. Shahar, S. Bergman, and M. Silberstein, “Activepointers: a case for software address translation on gpus,” *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 596–608, 2016.

[59] D. R. Engler, M. F. Kaashoek, and J. O’Toole Jr, “Exokernel: An operating system architecture for application-level resource management,” *ACM SIGOPS Operating Systems Review*, vol. 29, no. 5, pp. 251–266, 1995.

[60] J. Liu, J. Wu, and D. K. Panda, “High performance rdma-based mpi implementation over infiniband,” *International Journal of Parallel Programming*, vol. 32, no. 3, pp. 167–198, 2004.

[61] I. T. Association *et al.*, “Infiniband architecture specification release 1.2,” *http://www. infinibandta. org*, 2000.

[62] L. Zhao and D. Lie, “Is hardware more secure than software?” *IEEE Security & Privacy*, vol. 18, no. 5, pp. 8–17, 2020.

[63] B. Laurie, “How To Ruin A Perfectly Good Container,” https://medium.com/@benlaurie_18378/how-to-ruin-a-perfectly-good-container-d33250fca595.

[64] L. E. Olson, J. Power, M. D. Hill, and D. A. Wood, “Border control: Sandboxing accelerators,” in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2015, pp. 470–481.

[65] J. B. Dennis and E. C. Van Horn, “Programming semantics for multiprogrammed computations,” *Communications of the ACM*, vol. 9, no. 3, pp. 143–155, 1966.

[66] N. P. Carter, S. W. Keckler, and W. J. Dally, “Hardware support for fast capability-based addressing,” *ACM SIGOPS Operating Systems Review*, vol. 28, no. 5, pp. 319–327, 1994.

[67] E. A. Feustel, “The rice research computer: a tagged architecture,” in *Proceedings of the May 16-18, 1972, spring joint computer conference*, 1971, pp. 369–377.

[68] J. Woodruff, R. N. Watson, D. Chisnall, S. W. Moore, J. Anderson, B. Davis, B. Laurie, P. G. Neumann, R. Norton, and M. Roe, “The cheri capability model: Revisiting risc in an age of risk,” in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*. IEEE, 2014, pp. 457–468.

[69] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter, “Leaky cauldron on the dark land: Understanding memory side-channel hazards in sgx,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2421–2434.

[70] D. Lee, D. Jung, I. T. Fang, C.-C. Tsai, and R. A. Popa, “An {Off-Chip} attack on hardware enclaves via the memory bus,” in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[71] 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)*. usenix.org, 2017.

[72] C. Priebe, D. Muthukumaran, J. Lind, H. Zhu, S. Cui, V. A. Sartakov, and P. Pietzuch, “Sgx-lkl: Securing the host os interface for trusted execution,” *arXiv preprint arXiv:1908.11143*, 2019.

[73] S. Karandikar, H. Mao, D. Kim, D. Biancolin, A. Amid, D. Lee, N. Pemberton, E. Amaro, C. Schmidt, A. Chopra *et al.*, “Firesim: Fpga-accelerated cycle-exact scale-out system simulation in the public cloud,” in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2018.

[74] A. Akram, V. Akella, S. Peisert, and J. Lowe-Power, “Enabling design space exploration for risc-v secure compute environments,” 2021.

[75] K. Suzaki, K. Nakajima, T. Oi, and A. Tsukamoto, “Ts-perf: General performance measurement of trusted execution environment and rich execution environment on intel sgx, arm trustzone, and risc-v keystone,” *IEEE Access*, vol. 9, pp. 133 520–133 530, 2021.

[76] N. Weichbrodt, P.-L. Aublin, and R. Kapitza, “sgx-perf: A performance analysis tool for intel sgx enclaves,” in *Proceedings of the 19th International Middleware Conference*, 2018, pp. 201–213.

[77] M. Bailleu, D. Dragoti, P. Bhatotia, and C. Fetzer, “Tee-perf: A profiler for trusted execution environments,” in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019, pp. 414–421.

[78] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno, “Komodo: Using verification to disentangle secure-enclave hardware from software,” in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 287–305.

[79] K. Cheang, C. Rasmussen, D. Lee, D. W. Kohlbrenner, K. Asanovic, and S. A. Seshia, “Verifying risc-v physical memory protection,” in *SECRISC-V*, 2020.

[80] J. Zhu, R. Hou, X. Wang, W. Wang, J. Cao, L. Zhao, F. Yuan, P. Li, Z. Wang, B. Zhao *et al.*, “Enabling privacy-preserving, compute-and data-intensive computing using heterogeneous trusted execution environment,” *arXiv preprint arXiv:1904.04782*.

[81] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley, “Iso-x: A flexible architecture for hardware-managed isolated execution,” in *47th IEEE MICRO*, 2014.

[82] 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 *54th Annual IEEE/ACM MICRO*, 2021, pp. 199–211.

[83] R. Boivie and P. Williams, “Secureblue++: Cpu support for secure execution,” *IBM, IBM Research Division, RC25287 (WAT1205-070)*, pp. 1–9, 2012.

[84] “Protecting Register State with AMD SEV-ES,” https://developer.amd.com/sev/.

[85] “AMD SEV-SNP,” https://developer.amd.com/sev/.

[86] E. Feng, X. Lu, D. Du, B. Yang, X. Jiang, Y. Xia, B. Zang, and H. Chen, “Scalable memory protection in the penglai enclave,” in *15th USENIX OSDI*, 2021, pp. 275–294.