SoK: Understanding Designs Choices and Pitfalls of Trusted Execution Environments

Mengyuan Li
lmy@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

Yuheng Yang
yuhengy@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

Guoxing Chen
guoxingchen@sjtu.edu.cn
Shanghai Jiao Tong University
Shanghai, China

Mengjia Yan
mengjiay@mit.edu
Massachusetts Institute of Technology
Cambridge, Massachusetts, USA

Yinqian Zhang*
yinqianz@acm.org
Southern University of Science and Technology
Shenzhen, China

ABSTRACT

Trusted execution environment (TEE) is a revolutionary technology that enables secure remote execution (SRE) of cloud workloads on untrusted server-side computing platforms. Both commercial and academic TEEs have been proposed in the past few years, including Intel's SGX and TDX, AMD's SEV, ARM's CCA, IBM's PEF, and their academic counterparts built atop open-source RISC-V processors, such as Keystone, Sanctum, CURE, and Penglai. While great efforts from both sides have been made in developing a confidential computing ecosystem, the existence of server-side TEEs with drastically different designs and the presence of various known attacks have significantly increased the difficulty of understanding TEE designs and the reasons behind existing attacks.

This paper offers a structured analysis of the design choices of server-side TEEs, focusing on dissecting TEE designs and identifying their potential pitfalls. We introduce the TEE Runtime Architectural Framework (TRAF), a detailed framework that facilitates a thorough and methodical dissection of TEE designs by analyzing the high-level considerations made by TEE designs. A key aspect of TRAF's analysis is the reconfiguration of resource management in TEE designs, where the host OS used to have full control. By incorporating the Trusted Computing Base (TCB), TEE designs adopt different design choices on how to divide and coordinate tasks between the host OS and TCB to achieve security and effective management of computational resources. TRAF specifically investigates how common resources, such as CPU, memory, and I/O devices, are managed jointly by the TCB and host OS. This includes a focused study of factors that influence design choices, such as TCB size, performance, and efficiency. Furthermore, by examining existing vulnerabilities and attacks on TEEs, the paper further evaluates the security impact of varied design choices.

CCS CONCEPTS

• **Security and privacy → Security in hardware**.

KEYWORDS

Trusted Execution Environment, Confidential Computing, Secure Remote Execution

ACM Reference Format:
Mengyuan Li, Yuheng Yang, Guoxing Chen, Mengjia Yan, and Yinqian Zhang. 2024. SoK: Understanding Designs Choices and Pitfalls of Trusted Execution Environments. In *Proceedings of ACM ASIA Conference on Computer and Communications Security (AsiaCCS'24)*. ACM, New York, NY, USA, 17 pages. https://doi.org/XXXXXXX.XXXXXXX

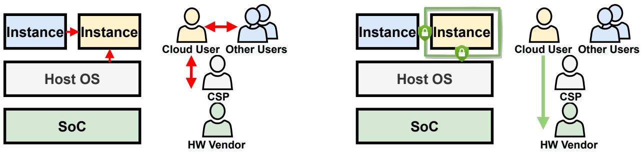
1 INTRODUCTION

In recent years, the IT industry has seen rapid growth in the cloud services market. Cloud customers rent computing power, in the form of execution instances (*e.g.*, VMs and containers) from CSPs and use the cloud for remote computing tasks. However, traditional cloud models require customers to fully trust CSPs. This trust allows CSPs to inspect and control user instances, as seen in Figure 1a. Such a demand for unreserved trust significantly hinders the adoption of cloud services for tasks involving sensitive data, especially for industries like finance, healthcare, and government.

A promising solution is confidential computing [20], which leverages server-side Trusted Execution Environments (TEEs) to protect sensitive workloads from untrustworthy CSPs. Enabled by hardware Trusted Computing Base (TCB) components, such as system-on-chips (SoCs) with memory management units (MMU) and memory encryption engines (MEE), TEEs provide a highly secure and isolated execution environment for *TEE instances.* This protected environment offers strong confidentiality and integrity guarantees for TEE instances, safeguarding them from privileged software or even direct physical attacks on the server. With a cryptographic root-of-trust (RoT) embedded in the SoC, TEEs also provide mechanisms for remote attestation to validate the authenticity of the hardware and the TEE instances. This advancement in TEE technology has significantly altered the trust dynamics within public

[*] Yinqian Zhang is affiliated with the Research Institute of Trustworthy Autonomous Systems and the Department of Computer Science and Engineering of Southern University of Science and Technology (SUSTech).

AsiaCCS'24, July 01–05, 2024, Singapore
© 2024 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX



(a) Traditional cloud model.

(b) Cloud model with TEE.

Figure 1: Cloud computation trust relationship comparison. The red arrows represent unconditional trust that could lead to potential attacks, while the green arrows represent trust.

clouds, positioning confidential computing as the *de facto* solution to protecting "data-in-use" in the cloud (as shown in Figure 1b).

Given the great benefits offered by TEEs, recent years have witnessed an increasing number of TEE production systems by major chip vendors, including Intel, AMD, and ARM, as well as academic TEE design proposals with open-source prototypes on RISC-V systems. These TEE designs vary in multiple aspects, including performance overhead, security properties, and programmability (i.e., easy to use). Despite being popular, TEEs have exhibited another concerning trend, with an increasing number of software and hardware attacks frequently catching the spotlight. We think that, with the thriving research efforts on TEE designs, the community could benefit significantly from the systematization of these efforts, which can help provide a big picture of the diverse TEE design landscape and identify promising research problems.

However, deriving a systematic view of the diverse TEE research landscape is challenging. A primary difficulty is that, given the abundance of TEE designs, characterized by their diversity in design and implementations, understanding the intricate details and underlying considerations of these complex but distinct designs can be a real challenge, especially for new researchers in the domain. Moreover, not all design choices and implementation details are equally important. A desired systematization should help people to grasp the architecture-level motivations that drive TEE design choices and logically reason about the trade-offs behind these choices. Thus, this paper provides a TEE deconstruction framework which focuses on the fundamental challenge of TEE designs:

> *How TEE design can safeguard resources used by TEE instances while also enabling an untrusted Operating System (OS) to manage computing resources in an effective way for CSP's responsibilities.*

By breaking down TEE designs from the perspective of resource management tasks of the OS, the framework thus provides a systematic and structured methodology to understand the complexities of TEE designs from a top-down approach.

TEE Runtime Architectural Framework (TRAF). TEE Runtime Architectural Framework is designed to assist newcomers to gain a comprehensive understanding of an unfamiliar TEE design. TRAF offers a structural approach based on common events and tasks (*e.g.*, CPU scheduling, memory management, *etc.*) in the OS, breaking down a complex TEE design into how the TEE design considers different events and resource management. For different events, a TEE design can choose to take direct control of the events or delegate them to the untrusted host OS for handling. TRAF classifies critical system management events into four modes. The four modes emphasize the coordinating relationship between the TEE instance, the untrusted privileged OS, and the trusted protection measures introduced by TEE (components within TCB). The adoption of different modes reflect an architectural overview of TEE design choices and is due to different trade-offs, such as security, performance, and efficiency. Through this high-level analysis, researchers can swiftly grasp the emphasis of TEE designs. Consequently, they can quickly gain a rough understanding of the general considerations and select specific areas of interest for further in-depth research. Furthermore, we hope that this deconstruction model will motivate researchers to systematically consider the security of TEE designs from the perspective of the impact of different OS events on TEE design.

The contributions of this work are summarized below:

- This paper introduces TRAF, a structured framework designed to dissect TEE designs and analyze the division and coordination of resource management tasks between the Trusted Computing Base (TCB) and the host OS.
- This paper leverages TRAF to perform an in-depth analysis of the high-level design choices governing how TEE designs handle common runtime tasks. The analysis reveals that most existing TEE designs exhibit similar design choices for most tasks. This paper also discusses implications stemming from design choices, including their impact on security, TCB size, and performance.
- This paper categorizes known attacks targeting TEEs and delves into a case study concerning the reasons behind existing attacks on AMD SEV and the corresponding countermeasures. It highlights that attacks stemming from TEE design flaws often result from certain resources still being managed by the untrusted host OS in some corner cases, where they lack adequate protection.

2 TEE IN A NUTSHELL

In this section, we first provide essential background information about TEEs. In addition, we provide an overview of a generalized TEE lifecycle, which mainly involves remote attestation and runtime protection.

2.1 Scope of Investigation

In this paper, we focus our attention on *hardware*-based server-side (*a.k.a.*, cloud) TEEs in general-purpose processors that support a *secure remote execution* (SRE) feature [80]. A TEE is considered hardware-based if hardware-backed techniques are used to provide the security guarantees of TEE instances running inside the protected environment [19]. SRE refers to a security feature that enables a secure execution of applications on remote, untrusted platforms. Specifically, a TEE design with the SRE feature must have the following security properties [80]: (1) *Secure Measurement:* It should provide a measurement of the underlying hardware platform and the initial states of the TEE instance that could be used for remote attestation; (2) *Confidentiality:* It should keep the inner data of the TEE instances confidential against software and physical (optional) adversaries; (3) *Integrity:* It should prevent unauthorized tempering of the code and data of the TEE instances.

Not included in our study are TEEs that do not support SRE or those implemented purely in software. For instance, TrustZone [6] is a well-known TEE that has been available on ARM processors for over fifteen years. However, as it does not by design for cloud platform and does not support secure measurement to allow remote users to verify the initial states of the TEE instances, we exclude TrustZone from our investigation. Also excluded are TEEs implemented entirely in software, such as Amazon AWS Nitro enclave [8] and Ant Group's HyperEnclave [42]. Software extensions of hardware-based TEEs like vSGX [102], Twinvisor [51] and Colony [97] are also excluded in this paper, because their security guarantees are built atop the underlying TEEs. In addition, we only focus on major server-level processor platforms, including Intel, AMD, ARM, and RISC-V. Embedded TEE designs are excluded from the scope of this paper due to their different use scenarios and diverse design considerations.

2.2 TEE Threat Model

Threat models. The adversary that a TEE design typically considers has the power of controlling the entire software stack of the underlying computing platform. Specifically, it is assumed that the adversary is able to (1) execute arbitrary instructions at the privileged levels; (2) launch, pause, and destroy TEE instances at his will; and (3) manage the software stack in the non-TEE world, including memory mapping, I/O devices, and CPU scheduling. In addition, many TEE designs also assume that the adversary could gain physical access to the computing devices [5, 29, 36, 44, 49, 83]. In the presence of a physical adversary, peripheral hardware components outside the SoC are considered untrusted, as they can be compromised via a variety of physical attacks, such as cold-boot attacks [31] and bus-snooping attacks [64]. However, attacks targeting availability, another key component of the CIA triad [72], are generally not considered within the threat model of commercial TEEs. This is because the computing platform, including both hardware and software, is managed by a potentially adversarial party. Terminating or crashing the TEE instance is the least an adversary could do. Consequently, Denial-of-Service (DoS) attacks are excluded in TEE's threat model. Moreover, microarchitectural side-channel attacks are also typically excluded from the threat model commercial TEEs [3, 5, 22, 36].

Trusted computing base (TCB). The TCB of TEEs includes all hardware and software components that are considered trusted and are involved in the establishment of TEE's security foundation. Specifically, the TCB of TEEs can be further classified as *Manufacturer TCB* and *ISV TCB*. Manufacturer TCB includes both hardware and software components that are provided by the TEE manufacturer. For instance, in Intel SGX [22], Manufacturer TCB includes manufacturer-provided hardware components, such as Instruction Set Architecture (ISA), hardware-based memory encryption engine, microcode-based MMU extension, and also software-based architectural enclaves (*e.g.*, Quoting Enclave and Provisioning Enclave). ISV is short for Independent Software Vendor, which is the term coined by Intel to refer to software developers that utilize TEEs to build applications. ISV TCB, therefore, includes the software running inside the TEE-protected execution environment (*e.g.*, processes running inside enclaves or confidential VMs). Note that even though vulnerabilities inside the ISV TCB may lead to various attacks (*e.g.*, memory safety attacks [18] and thread safety attacks [91]), vulnerabilities within ISV TCB are usually out of the scope of a TEE design, as it is the ISV's responsibility to ensure a bug-free ISV TCB.

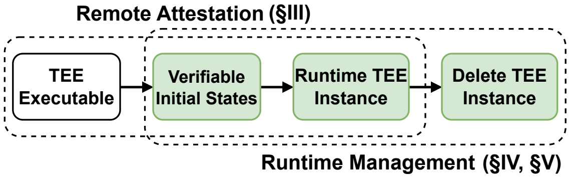

Remote Attestation (§III)
TEE Executable
Verifiable Initial States
Runtime TEE Instance
Delete TEE Instance
Runtime Management (§IV, §V)

Figure 2: Generalized TEE lifecycle.

TCB size is a common factor used to reflect the security of a TEE design. Generally speaking, a smaller TCB size indicates fewer lines of code or hardware implementations, which in turn suggests a reduced likelihood of vulnerabilities. However, comparing the security of two TEE designs based on the TCB size is difficult, even between VM-based TEE (e.g., AMD SEV) and process-based TEE (e.g., Intel SGX). For example, it is generally assumed that the ISV TCB of a VM-based TEE is larger than that of a process-based TEE. However, as we see more research efforts to build smaller TEE OS with memory safe programming languages (*e.g.*, Rust), it is unreasonable to draw such a conclusion. Regarding the Manufacturer TCB, as VM-based TEEs generally need to handle VM-specific events (such as privileged instructions or I/O), their Manufacturer TCB may be larger or more complex. However, it is still hard to directly compare the Manufacturer TCB size between two TEE designs, as most commercial TEEs are closed-sourced.

2.3 Generalized TEE Lifecycle

As shown in Figure 2, TEE designs and the lifecycle primarily involve two parts: remote attestation and runtime management.

- ***Remote attestation.*** Remote attestation generates a verifiable report detailing the initial states of the TEE instance, such that a remote user could verify the authenticity and integrity of both the Manufacturer TCB (the execution environment) and ISV TCB (the TEE software). The initial states consist of a cryptographic hash of the deployed software binary, as well as the configuration of the TEE's hardware and software. Most TEEs follow a similar remote attestation procedure to generate the attestation report with assistance from the Manufacturer TCB.
- ***Runtime management.*** Starting with verified initial states, in runtime management, TEE designs primarily focus on two aspects: *efficient resource management* and *security*. In terms of *efficient resource management*, TEE designs must align with the resource management requirements of CSPs. This ensures efficient administration of server resources, including CPU, memory, and I/O devices, which are utilized by both TEE instances and non-TEE components during runtime. In terms of *security*, TEE designs must protect the confidentiality and the integrity of TEE instances under TEE threat model (Section 2.2).

Most TEE designs typically follow similar steps in Remote Attestation. Therefore, we briefly summarize the steps for performing remote attestation and verifying the initial states of TEE instances in Section 3. However, approaches to runtime management and protection vary more broadly across TEE designs. Hence, this paper mainly focuses on how different TEE designs coordinate the TCB with the untrusted host OS to provide secure and effective runtime management. In Section 4 and Section 5, we will present a systematic framework to analyze how cloud TEE designs achieve runtime management.

3 REMOTE ATTESTATION

In this section, we briefly introduce how most TEE follows a general remote attestation procedure to ensure verifiable TEE initial states.

3.1 General Remote Attestation Procedures

Most TEEs follow the same remote attestation procedure:

Step1. The TEE instance itself or the TEE instance's owner (the entity the loads the TEE instance) initiates a remote attestation request to a trusted attestation supervisor. The trusted attestation supervisor is usually within the Manufacturer TCB, which can be the underlying hardware, a secure co-processor or a privileged TEE instance released by the TEE manufacturer;

Step2. The attestation supervisor generates and signs a verifiable attestation report. This report includes critical information about the TEE instance, such as a secure measurement of its initial states, platform details, and configurations;

Step3. TEE instance users then verify the attestation report before transmitting any secret data to the TEE instance.

The key technical components of remote attestation include (1) a chain of trust, employed to authenticate the trustworthiness of both the attestation report and the attestation supervisor, and (2) The establishment of a secure measurement mechanism capable of accurately representing the initial states of TEE instances.

3.2 Chain of Trust

The root of trust is usually a root signing key owned by the TEE manufacturer. The manufacturer provisions a specific key, named *attestation key*, to each TEE SoC and uses the root signing key to endorse this attestation key. The attestation key is then used by an attestation supervisor to sign attestation requests from TEE instances. The attestation supervisor can vary according to different TEE design, but is usually within the Manufacturer TCB. Common attestation supervisors could be (1) the TEE hardware, or (2) a privileged TEE instance provided by the manufacturer. Compared with implementing the signature algorithm in hardware, leveraging software-based implementation (*i.e.*, a privileged TEE instance) enables more advanced signature algorithms (*e.g.*, EPID adopted by Intel SGX could protect the privacy of the TEE platforms). On the downside, the privileged TEE instance becomes a naturally target of the adversary, which might have a larger attack face. For example, several speculative-execution attacks targeting the Intel's privileged enclave (*i.e.*, Quoting Enclave) have successfully obtained the attestation key [16, 81], thus breaking the security guarantees.

After obtaining the signed attestation report, verifiers require the corresponding verification key for the attestation key to verify the report. When the manufacturer opts not to make verification keys public, or to release them to limited third parties (*e.g.*, cloud providers or data centers), the manufacturer or the third parties need to run some attestation service to perform the verification on behalf of the TEE user. Consequently, in such cases, TEE users must place additional trust in the attestation service. Once the verification process is successfully completed, TEE users can use the public key provided by the TEE instance, which is included in the attestation report, to establish a secure channel with the TEE instance. Through this secure channel, TEE users can securely provision data or delegate computation tasks to the TEE instance.

3.3 Secure Measurement

Secure measurement aims to deterministically generate a fix-length measurement from the content of the TEE to identify the TEE instance. Existing TEE designs [9, 23, 33] usually use the cryptographic hash of the initial content of the TEE instance as the measurement. It is usually assumed that the TEE user knows the initial content of the TEE instance in advance so that she could derive an expected value of the measurement. As long as the measurement of a communicating TEE instance matches the expected value, the TEE user is convinced that the TEE instance runs the expected software. As for the initial content, process-based TEEs usually measure the initial code and data of the applications while VM-based TEEs usually measure a secure BIOS which is the first piece of software loaded into the VM and rely on the BIOS to check the trustworthiness of the OS and applications loaded later.

The security of the measurement lies in that the adversary could not launch two TEE instances with different contents (*e.g.*, one malicious TEE instance and one victim TEE instance) but the same measurement. By using cryptographic hash functions, it is negligible that the adversary could find two different messages resulting in the same hash value. Hence, the key point here is that the attestation supervisor needs to ensure that all necessary information about the TEE instance's content (*e.g.*, virtual addresses, access permissions) is properly measured, and overdue data copies in micro-architectures are emptied. Otherwise, potential attacks could be launched to bypass the remote attestation. For example, Wilke *et al.* [96] discovered that AMD SEV-ES does not properly measure the virtual addresses of the secure BIOS code to be loaded into the VM, and thus could manipulate the layout of the secure BIOS without changing the measurement. The manipulated BIOS could then launch arbitrary malicious code and bypass the remote attestation.

4 TEE RUNTIME ARCHITECTURAL FRAMEWORK

In the rest of the paper, we mainly focus on the analysis of TEE's runtime management and protection. This section aims to provide a systematic framework for deconstructing TEE designs.

4.1 TRAF Overview

TEE Runtime Architectural Framework (TRAF) deconstructs a TEE design by taking advantage of the classification of common events and tasks from the perspective of operating system (OS). TRAF specifically concentrates on the events that frequently occur in cloud TEE runtime, *a.k.a.*, runtime management tasks (as shown in Figure 3), and further extends the analysis to assess how TEE designs safeguard these events.

Specifically, TEE designs revolve around the dual goals of efficient resource management and security in TEE runtime, while aiming to avoid excessive TCB expansion. Rather than incorporating the entire OS stack (*e.g.*, the CPU scheduler, page table management, and I/O drivers) into the TCB, TEE designs must also align with the resource management requirements from the untrustworthy CSP

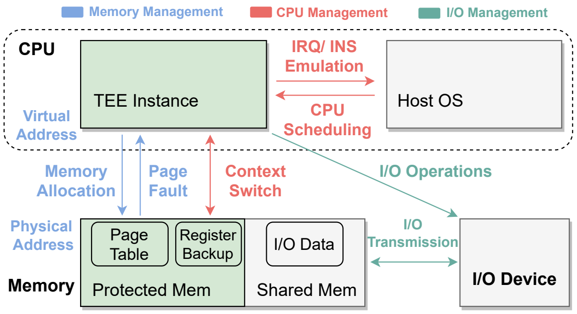


Figure 3: Common events and resource management tasks (CPU, memory, and I/O) for TEE instances.

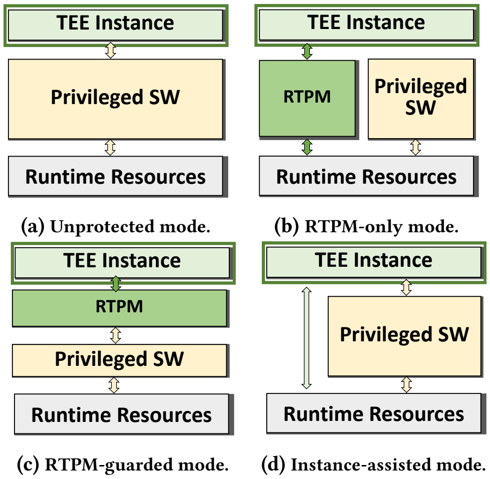
(*a.k.a.* host OS). This results in the design challenges that the TEE runtime is managed through coordination between the untrusted host OS and the Manufacturer TCB. How common runtime management tasks are split between the two leads to distinct high-level design choices, which are chosen by various TEE designs.

TRAF first categorizes high-level design choices into four modes in Section 4.2, where each mode indicates whether and how a TEE design takes care of the event. We then show how each TEE design chooses from these modes for different runtime tasks, including CPU virtualization (Section 4.3), memory management (Section 4.4), and peripheral resource management (Section 4.5), ***as what is usually covered in an OS textbook*** [7]. A timeline view of how design choices evolve over time is presented to help show the trend in Figure 5.

4.2 High-level Design Choices

To what extent does a TEE design still delegates the ability of managing runtime resources to privileged software is a crucial factor in server-side TEE designs. Specifically, to secure TEE instances, TEEs perform some runtime management tasks inside their Manufacturer TCB. These tasks can be conducted by extra SoC components (*e.g.*, AMD secure co-processor [3]), CPU extensions (*e.g.*, Intel TDX ISA [36]), or software components (*e.g.*, Intel TDX module [36] and SGX quoting enclaves [22]). We coin a term *TEE Runtime Protection Module* (*RTPM* for short) to refer to these Manufacturer TCB components that involve secure runtime management. In a TEE design, the RTPM performs resource management in collaboration with the privileged but untrusted host OS in one of the four modes:

Unprotected mode. As depicted in Figure 4(a), in the unprotected mode, the privileged software manages the runtime resources directly, and the TEE design fully leverage the untrusted software for resource management. When choosing this mode, privileged software directly manages the hardware resources allocated to the TEE instances. One example is about scheduling CPU resources to TEE instances such as setting up CPU affinity. Note that ISV TCB may play a role in introspection or attack detection in some cases. However, this is possibly the behavior of the ISV itself. Therefore, it should also be attributed to unprotected mode if the Manufacturer TCB does not participate. An example is instruction emulation on SEV environment, where the guest VM (ISV TCB) may additional check the instruction return value in some scenarios.



(a) Unprotected mode. (b) RTPM-only mode.

(c) RTPM-guarded mode. (d) Instance-assisted mode.

Figure 4: Four modes for TEE runtime management. (a) Untrusted privileged OS fully manages runtime resources; (b) RTPM fully manages runtime resources; (c) Untrusted privileged OS configures runtime resources and RTPM performs additional security checks; (d) TEE instance itself performs runtime resource management.

RTPM-only mode. As shown in Figure 4(b), in RTPM-only mode, the interactions between the TEE instance and its associated runtime resources are exclusively managed by the RTPM and the privileged software only handles resources for other non-TEE components. TEE designs may opt for RTPM-only mode when handling security-critical but simple runtime management tasks, such as context switches where registers are securely saved and restored by the RTPM. However, performing complex tasks with this mode may significant increase the size of RTPM and thus the TCB size.

RTPM-guarded mode. Figure 4(c) illustrates the RTPM-guarded mode. In this mode, runtime management tasks are coordinated between the privileged software and the RTPM, usually with the former handling resource management and the latter focusing on security validation. The RTPM-guarded mode is an appealing design choice for complex runtime management tasks, such as memory management. For example, in AMD SEV-SNP, the privileged software performs nested page table maintenance and fault handling, where as the RTPM validates proper configuration of memory encryption key or isolation.

TEE Instance-assisted mode. Figure 4(d) describes the TEE instance-assisted mode, where the TEE instance itself needs to directly manipulate some runtime resources. Without the RTPM or untrusted host OS working as the agent for the TEE instance, the instance has to perform part of the management task itself. Different from the previous two modes, the instance-assisted mode does not offer the transparency of global runtime resources to the TEE instance and can hurt the usability of the system. Therefore, very few TEE designs take this approach for physical resources. One such example is that Keystone lets the TEE instance itself handle page faults.

4.3 CPU Virtualization

CPU virtualization provides an abstraction of the CPU resources to TEE instances. We focus mainly on events that involve interaction between TEE instances and the untrusted host, including CPU scheduling, context switch, and interrupt and instruction emulation (for VM-based TEE).

4.3.1 CPU Scheduling. In a multiprogramming system, CPU scheduling selects among idle processes and decides the next process to occupy the CPU core. The privileged software in charge of making such decisions is called the CPU scheduler. Similarly, TEE instances are also managed as processes/VMs and share CPU resources with non-TEE components. Most TEEs exclude the CPU scheduler from the RTPM and adopt the unprotected mode for CPU scheduling, where the scheduling decision is made solely by the scheduler in the untrusted OS. This choice is motivated by two key reasons: (1) CSP require the ability to efficiently manage and determine the scheduling of TEE instances according to their specific needs and policies. Adopting a RTPM-guarded mode in CPU scheduling may introduce significant performance degradation. (2) Due to the complexity of CPU scheduling, placing the functionality of a CPU scheduler into the RTPM will significantly increase the TCB size [11].

Scheduling algorithms that could be applied in a scheduler, such as Round Robin (RR) or Completely Fair Scheduling (CFS) used in Linux, are also chosen at the need of CSPs. The scheduler relies on interrupts (*e.g.*, timer interrupts and I/O interrupts) to force context switches between processes. Thus, the choice of unprotected mode for CPU scheduler leverages the untrusted OS the ability to interrupt a TEE instance at his will, which may result in interrupt-based controlled-channel attacks, as discussed in the implications of design choices (Section 5).

> **Observation:** Most TEEs adopt the unprotected mode for CPU scheduling to enable efficient resource management by CSPs and reduce their TCB size.

4.3.2 Context Switch. During context switches, the registers used to represent the context of a TEE is saved to the memory. To protect the integrity and confidentiality of the register values, most TEEs adopt the RTPM-only mode for context switch. For example, in the legacy AMD virtualization architecture, the hypervisor assists the saving and restoring of some registers of a VM [43]. In AMD SEV-SNP, however, the entire procedure is merged together as single atomic hardware instructions (`VMRUN` abd `VMEXIT`), and the register values are stored in an RTPM-protected memory region. For TEEs based on open-sourced RISC-V, the trusted secure monitor running in the machine mode traps and takes over the context-switch procedure to ensure its confidentiality and integrity.

Besides updating register values, RTPM may optionally clean data stored in some microarchitectures (*e.g.*, TLB and cache) during context switch. For example, a complete TLB flush is usually required during the context switch to avoid reusing TLB entries of other processes. Some TEEs (*e.g.* Intel SGX) may also explicitly and partially flush cache as well to prevent speculative execution attacks [92] and cache side-channel attacks. RTPM in some TEEs (*e.g.*, Keystone and Penglai) choose to physically partition cache ways and switch cache ways during context switch to prevent cache side-channel. For performance concerns, some TEEs (*e.g.* AMD SEV and Intel TDX) tag TLB entries with a security domain label and switch this label during context-switch to avoid the total TLB flush. To do so, RTPM uses an identifier of the current executor to label each TLB entry. With such a TLB tag, the TLB entries left by the previous domains will not pollute each other.

> **Observation:** Most TEEs choose the RTPM-only mode for context switch. AMD SEV is an exception as it follows the unprotected mode, but the subsequent SEV-ES/SEV-SNP switch to the RTPM-only mode.

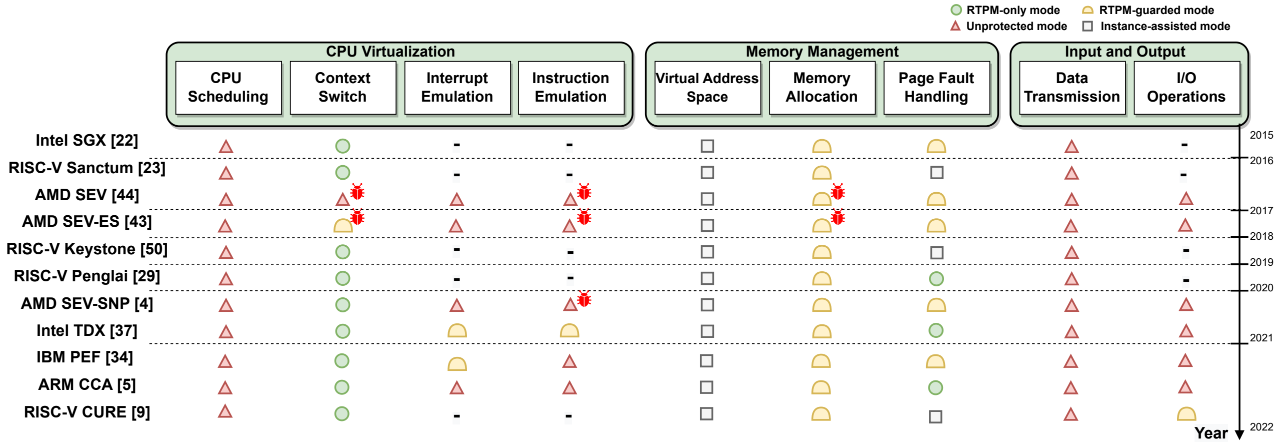
4.3.3 Interrupt and Instruction Emulation. In VM-based TEE, internal interrupts and exceptions work in an emulated way, which can be achieved by emulating the states of APIC-related MSR in the VM control block. Some TEE designs don't provide additional protection for all interrupts and exceptions, such as AMD SEV, which still uses the interrupt and exception mechanism used in non-TEE virtualization environment. This means that the host can arbitrarily interrupt the VM directly, and when an exception occurs, it is the responsibility of the host to handle or redirect it. Some TEE designs use the RTPM-guarded mode, such as Intel TDX and IBM PEF, which leverage RTPM to trap, handle some security-related interrupts and exceptions.

Similarly, certain instructions related to the physical state, such as `CPUID`, `RDMSR`, `RDTSCP`, *etc.*, require emulation in an virtualized environment. In the traditional non-TEE virtualization environment, these instructions are emulated by triggering `VMEXIT`s, and letting the hypervisor update the corresponding registers. VM-based TEEs, including AMD SEV, SEV-ES, and SEV-SNP, inherit similar mechanisms (unprotected mode), where the VM or RTPM directly exposes the registers associated with the instructions to the host. The host then provides the simulated results and passes them directly to the VM. The reports of AMD SEV-SNP [3] introduced that a malicious host can induce a VM to use a vulnerable cryptographic implementation, or cause different VM behaviors, by providing incorrect `CPUID` return values, which is a significant violation of the integrity of the TEE execution flow. Although the VM can add additional checks internally to roughly check the return values, the Manufacturer TCB does not play a supervisory role throughout the process.

> **Observation:** For VM-based TEEs, instruction emulation and interrupt emulation are essential in existing virtualization technologies. However, since the untrusted hypervisor must be involved, TEEs usually operate in unprotected mode or RTPM-guarded mode.

4.4 Memory Management

Memory management aims to optimize the utilization of physical memory and provide isolation of the memory used by different processes. Three key elements are essential in most modern OS for memory management: the management of virtual memory (Section 4.4.1), the allocation of physical memory (Section 4.4.2), and the handling of page faults (Section 4.4.3). In addition, we include a discussion of memory encryption (Section 4.4.4), which is widely used to protect TEE's memory against physical attacks. We exclude the discussion of internal memory management of VM-based TEEs, as it is usually completely managed by the VM itself.



	CPU Virtualization				Memory Management			Input and Output	
	CPU Scheduling	Context Switch	Interrupt Emulation	Instruction Emulation	Virtual Address Space	Memory Allocation	Page Fault Handling	Data Transmission	I/O Operations
Intel SGX [22]	△	○	-	-	□	◠	◠	△	-
RISC-V Sanctum [23]	△	○	-	-	□	◠	□	△	-
AMD SEV [44]	△	△🐞	△	△🐞	□	◠🐞	◠	△	△
AMD SEV-ES [43]	△	◠🐞	△	△🐞	□	◠🐞	◠	△	△
RISC-V Keystone [50]	△	○	-	-	□	◠	□	△	-
RISC-V Penglai [29]	△	○	-	-	□	◠	○	△	-
AMD SEV-SNP [4]	△	○	△	△🐞	□	◠	◠	△	△
Intel TDX [37]	△	○	◠	◠	□	◠	○	△	△
IBM PEF [34]	△	○	◠	△	□	◠	◠	△	△
ARM CCA [5]	△	○	△	△	□	◠	○	△	△
RISC-V CURE [9]	△	○	-	-	□	◠	□	△	◠

Figure 5: Design choices used in representative TEEs. The bug symbol indicates there exists known attacks.

4.4.1 Virtual Memory Management. Virtual memory here refers to the address space seen directly by the TEE instances. In process-based TEEs, this is the virtual address space of the hosting process of the TEE instance; in VM-based TEEs, this refers to the guest physical address used by the VM (which is later translated to system physical address used by the main memory through a nested page table). In either case, the TEE virtual memory is managed in the instance-assisted mode. For example, VM-based TEEs allow the TEE instance to create page tables inside the TEE and add another indirection layer of virtual memory by itself. Process-based TEEs may not be allowed to create their own page tables inside the TEE, but TEE designs usually do not restrict TEE instances from arranging their private virtual address layout.

Observation: Most TEEs adopt the instance-assisted mode for virtual memory management.

4.4.2 Physical Memory Allocation (Access Control). Memory allocation allocates a set of physical addresses for TEE instances and initiates their mapping with virtual addresses. Memory allocation can take places statically when launching the TEE instances or dynamically via demand paging [29]. Most TEEs using RTPM-guarded mode for memory allocation follow two steps: (1) The untrusted OS first selects free physical pages from the main memory, optionally inside a region reserved for TEE, and (2) the RTPM performs additional checks to ensure security (*a.k.a.*, access control).

The mapping information (between the address space seen by the TEE instances and the system physical memory address space) needs to be guarded by the RTPM to ensure that private physical pages owned by a TEE instance are never mistakenly shared with others. Different TEEs choose different mechanisms to implement such security guarantee. For example, some TEEs (*e.g.*, AMD SEV-SNP, Intel TDX, Keystone) choose to perform ownership checks at the end of an address translation, such that the access permission check can be performed before the mapping information is cached into the TLB. Others perform such checking when initializing the page tables. For example, the secure monitor in RISC-V Penglai ensures the correctness of all page tables when updating page tables and therefore can omit access permission checks during page table walks. By ensuring no unauthorized individuals could have an illegal page table entry (*e.g.*, untrusted host has a mapping pointing to an enclave), Penglai provides access control with the help of strict memory allocation check.

The way in which the RTPM provides such check may also be different. For example, Intel SGX and AMD SEV-SNP rely on some metadata records (Enclave Page Cache Map (EPCM) for Intel SGX, and Reverse Map Table (RMP) for AMD SEV-SNP) to check the correctness of memory allocation and the permission of a memory access. However, AMD SEV and SEV-ES relies on indirect checks enabled by memory encryption to guard memory allocation and access (*e.g.*, an attacker cannot get the plaintext of other TEE instance's memory because the attacker will be forced by RTPM to use a wrong key for the decryption [53]).

Observation: Most TEEs adopt the RTPM-guarded mode for physical page allocation, which enforces security validations before memory accesses (either during page table update or address translation).

4.4.3 Page Fault Handling. The page table entries (PTEs) inside system-level page tables (*i.e.*, nested page tables for VM-based TEEs and page tables for process-based TEEs) that translate the virtual memory address to the system physical memory address play a critical role in regulating page fault exceptions. Existing TEE designs diverge in their choices of protecting page fault handling, with multiple modes being adopted. These protection mechanisms must ensure that memory allocation information remains uncompromised during fault handling to prevent malicious tampering.

RTPM-guarded mode. Some TEEs (*e.g.*, AMD SEV-SNP and Intel SGX) rely on untrusted privileged software to maintain the system-level page tables, but the RTPM performs permission checking at runtime to ensure the memory mapping information is still correct (*e.g.*, after page fault handling and page table walks). Using the RTPM-guarded mode can reduce the workload and TCB size of the RTPM, but as RTPM does not prevent the host from setting/unsetting the present bit of the PTEs, it opens up attack vectors for controlled-channel attacks (as discussed in Section 5.1).

RTPM-only mode. Some TEEs (*e.g.*, Intel TDX and RISC-V Penglai) choose to manage memory mapping in the RTPM-only mode, where the memory regions of page tables are only writable by the RTPM and page faults are also handled by the RTPM. The main difference between the RTPM-guarded mode and the RTPM-only mode is whether the untrusted OS is allowed to update the page table directly during runtime. Note that ARM CCA and Intel TDX have a separate nested page table maintained for VM's shared memory region, which is protected in the RTPM-guarded mode. Thus, ARM CCA and Intel TDX can also be categorized as a hybrid mode.

TEE instance-assisted mode. Some TEEs assign the task of maintaining the page tables to the TEE instance. For example, TEE instances in Keystone and CURE have their own runtime system, which maintains its own page table and handles its page faults. Although an ownership check will eventually be performed by the RTPM before a memory address is accessed, we regard this type as instance-assisted mode as the TEE instance has the ability to directly configure physical memory. TEE instance-assisted mode can also mitigate exception-based controlled-channel attacks.

> **Observation:** TEEs choose different modes for handling page faults, leading to varying security implications.

4.4.4 Memory encryption. When a physical attacker is also considered in the TEE threat model, all hardware devices other than SoC are not trusted. Thus, memory encryption seems to be the only way to protect data stored in the physical memory. When memory encryption is enabled, the RTPM includes either a software or hardware memory encryption engine (MEE) responsible for encrypting outbound data from the SoC and decrypting inbound data. The memory encryption keys are supposed to be managed and accessible only by RTPM, ensuring their security against any attackers during their lifecycle.

Unlike TEE designs with small memory sizes that prioritize memory encryption integrity and offer freshness support (*e.g.*, RISC-V Keystone and Intel SGX v1), most TEE designs supporting larger memory sizes tend to either remove such protections (*e.g.*, AMD SEV, Intel TDX, and Intel SGX v2) or make them optional (*e.g.*, ARM CCA). This is primarily driven by the substantial performance overhead associated with maintaining integrity or freshness metadata, leading many designs to forego integrity protection, which may introduce potential risks of memory replay attacks.

4.5 I/O Management

I/O are necessary for TEE instances to store data persistently or to transmit data with an external party. Process-based TEEs have data transmission with the non-TEE world through shared memory. VM-based TEEs also perform I/O operations (*e.g.*, I/O instructions, or DMA) with virtual I/O devices through shared memory.

Most existing VM-based and process-based TEE designs choose the unprotected mode for I/O traffic, allowing the privileged software to intercept or manipulate the I/O traffic. For instance, Intel SGX SDK allows enclaves to perform data transmission using OCalls (*i.e.*, temporarily exit the enclave to call an outside function), which are handled by untrusted software. VM-based TEEs, such as AMD SEV, ARM CCA, and Intel TDX, support I/O operations using software-emulated I/O devices. When memory encryption is enabled, emulated I/O devices cannot directly access VM's private memory. Thus, a shared memory region is used to simulate Direct Memory Access (DMA) operations. To support programmable I/O instructions (PIO), the TEE OS, like the Linux kernel adapted for AMD SEV-ES, uses a special handler to trap privileged I/O instructions, place the I/O data into a shared region, and trigger an on-demand context switch to the hypervisor to complete the emulation of the I/O instruction.

> **Observation:** Most TEE designs utilize the unprotected mode for I/O data transmission and I/O operations. The confidentiality and integrity of I/O operations is protected through encryption in the application layer.

Towards trusted I/O. Under the unprotected mode, the confidentiality and integrity of TEE's I/O operations solely relies on software-based encryption mechanisms (*e.g.*, TLS/SSL for network communication and full-disk encryption for disk I/O), which could introduce non-negligible performance overhead. To reduce such performance overhead, Intel recently introduced a white paper on hardware-supported trusted I/O for TDX v2.0 [38]. Specifically, I/O devices themselves need to include a trustworthy logic unit (*e.g.*, Device Security Manager in TDX 2.0), which is responsible for building a secure channel between TEE instances and I/O devices. The I/O traffic between I/O devices and the TEE instances are encrypted and integrity protected, which can be enabled by Integrity & Data Encryption feature in PCIe 5.0 standard [39]. The I/O devices needs to support features like Single Root I/O virtualization (SR-IOV) to ensure one-to-one mapping between TEE instances and I/O devices or virtual I/O functions [38].

5 UNDERSTANDING TEE DESIGNS USING TRAF

The results in Figure 5 indicate that TEE designs exhibit similar trends and considerations (*a.k.a.*, design choices) in most tasks. However, divergences also arise among different TEEs in certain tasks, such as page fault handling and instruction emulation. In this section, we will delve into the potential implications that arise from the selection of different protection modes.

5.1 Implications of Design Choices

Different mode choices will bring implications for security, TCB size, and performance, which are enumerated as below:

- *Unprotected mode.* Choosing unprotected mode can lead to the corresponding task or resources being completely traceable or controlled by the untrusted host. However, due to CSPs' requirements for global resource management, most cloud TEEs allow the untrusted OS to handle these resource-related tasks, such as CPU scheduling and I/O routing, in an unprotected manner. The unprotected mode generally offers better performance, as it avoids additional privilege switches between the untrusted OS and RTPM. In the case of CPU scheduling, adopting a RTPM-guarded mode is nearly impossible, as requiring RTPM validation for every scheduling operation would significantly degrade CPU performance.

  However, the drawbacks of the unprotected mode are evident, as it increases the attack surface. For instance, adopting an unprotected mode for CPU scheduling allows adversaries to suspend TEE instance execution through interrupts or configure CPU affinity for TEE instances. Such interrupt-based controlled-channel attacks [55, 83] can be combined with microarchitectural side channels to perform fine-grained secret exfiltration [12, 24, 62, 77, 84]. Recent TEE designs have tried to mitigate such attacks, by restricting how interrupts can be injected (SEV-SNP [3]) or handling some interrupts by the TEE instances themselves (CURE [9]). Moreover, for I/O events, software-based encryption for all I/O traffic becomes a necessary countermeasure that needs to be performed by TEE instances themselves.

- *RTPM-only mode.* RTPM-only mode is considered the most secure mode because all operations are performed within TCB. For highly confidential operations, such as context switch, most TEEs adopt the RTPM-only mode. The lack of sufficient protection in AMD SEV and SEV-ES (using unprotected or RTPM-guarded modes) has resulted in the corresponding leakage during context switch [32, 53, 56, 93]. On the other hand, placing all operations within RTPM may significantly increase the TCB size and is not suitable for complex tasks. For instance, in the design of AMD SEV/ES/SNP, the primary RTPM is the security co-processor. However, it is impractical to include page fault handling within this security co-processor due to its limited capabilities. Delegating complex tasks to RTPM in such scenarios would severely impact performance. In cases like this, only software RTPMs, such as the trusted module in Intel TDX or the security monitor in RISC-V Penglai, can efficiently manage relatively complex tasks.
- *RTPM-guarded mode.* RTPM-guarded mode strikes a balance between the security advantages and the potential increase in TCB size associated with the RTPM-only mode, as well as the resource management capabilities offered by the unprotected mode. However, RTPM-guarded mode does have three drawbacks: Firstly, it may still result in side-channel information leakage. For example, TEEs that adopt the RTPM-guarded mode for page fault handling, such as Intel SGX and AMD SEV, will suffer from page fault controlled-channel attacks [32, 54, 65, 66, 67, 79, 85, 89, 93, 98]. By manipulating PTE (*e.g.*, clear the "present" bit), the adversary can maliciously trigger page faults, which allow the adversary to observe the memory access patterns of the TEE instances at the page granularity. Secondly, RTPM-guarded mode typically involves privilege switches between the untrusted OS and RTPM, which may result in worse performance. For instance, in the case of RISC-V Penglai, when updating the page table, the security monitor inside RTPM needs to traverse all existing page tables before approving the update request. This design choice has been identified as one of the primary performance bottlenecks in Penglai. Thirdly, the implementation of the RTPM-guarded mode may overlook certain exceptional scenarios. For instance, in the case of SEV-ES where RTPM-guarded mode is chosen for context switching, there is a potential for TLB misuses when the untrusted host deliberately manipulates CPU affinity in some corner cases [56].
- *Instance-assisted mode.* A few TEE designs allow instances to directly assist or participate in certain resource management tasks. The adoption of these designs is driven by two main reasons: (1) the corresponding resources may be virtual, or (2) the TEE designs aim to avoid involving untrusted hosts and preventing the leakage of additional information. However, it is crucial to exercise caution when adopting the Instance-assisted mode in TEE designs, as TEE users may not always employ the correct strategies in compliance.

	Cause	Attack Surface	Vendor	Modes
CPU	Context Switch	Unprotected Regs [32]	SEV Only	Unprotected
	Context Switch	Unprotected Regs [93]	SEV Only	Unprotected
	Context Switch	TLB Misuse [56]	SEV/ES	RTPM-guarded
	Context Switch	Unauthenticated Encryption [53]	SEV/ES	RTPM-guarded
	Instruction Emulation	Debug Registers [3]	SEV/ES/SNP	Unprotected
	Instruction Emulation	Untrusted CPUID [3]	SEV/ES/SNP	Unprotected
MEM	Memory Allocation	Unprotected NPT [66]	SEV/ES	RTPM-guarded
	Memory Allocation	Unprotected NPT [65]	SEV/ES	RTPM-guarded
	Memory Allocation	Unprotected NPT [53]	SEV/ES	RTPM-guarded
	Memory Allocation	Unprotected NPT [67]	SEV/ES	RTPM-guarded

Table 1: Vulnerable design flaws.

Malicious TEE users may also attempt to exploit the instance-assisted mode, resulting in adverse effects. For example, if the entire address translation is performed inside the TEE, a malicious TEE instance can access arbitrary memory pages. Additionally, if interrupt handling is completely controlled by the TEE, a TEE instance could refuse to relinquish CPU resources, thereby making them unavailable to the host.

5.2 Security Implications: Known Design Flaws

To delve deeper into the security implications of various design choices, we compile a comprehensive list of known attacks against TEEs (Table 2 in Appendix A). There are numerous types of attacks targeting TEE, including *vulnerable system designs*, *microarchitectural side channels*, *hardware flaws*, *speculative execution vulnerabilities*, and other *microarchitecture flaws*. The majority of existing attacks fall into the categories of side-channel attacks, which are explicitly pointed out to be excluded in most commercial TEE's threat model [22, 44]. Our paper primarily focuses on the protection of TEE runtime and the security implications of how resources are managed. Therefore, in this section, we specifically delve into *vulnerable system designs* during TEE runtime. All other known attacks with different types will be discussed in Section 6.

Attacks due to*vulnerable system designs* arise from TEE design that fail to achieve their security goals in specific corner cases. Existing attacks often exploit inconsistent behavior in these corner cases and compromising the overall security of the TEE. Notably, the unprotected mode or RTPM-guarded mode are more prone to design flaws, as the untrusted host can influence the corresponding tasks. For example, in AMD SEV, the register values are not protected during context switches [32, 93], which gives attacker the time window to alter their values and breach the integrity of the VM instance. Similarly, lack of enough protection (RTPM-guarded mode) towards the nested page table (NPT) [65, 66, 67] and TLB [56] also leads to security breaches. All known attacks resulting from vulnerable design are listed in Table 1. Attacks resulting from microarchitecture flaws, such as insecure memory encryption engines [27, 94] or special instructions [99] that can bypass TEE isolation, can also be considered as stemming from vulnerable TEE designs to some extent. This is because these microarchitecture designs overlook corner cases related to TEE. However, these attacks are primarily attributed to flaws in the manufacturer's TCB itself, rather than issues within the system design and runtime management tasks. Therefore, we categorize them separately as a distinct class of attacks.

Furthermore, it is noteworthy that the majority of known attacks stemming from vulnerable system designs are primarily associated with the SEV series. This can be attributed to SEV being an early design for confidential VMs, which may have overlooked certain corner cases during its development. This discovery is valuable for our analysis of TEE designs, as it allows us to examine the evolution of the three versions of SEV (SEV, SEV-ES, and SEV-SNP) and the mitigations implemented to address existing attacks, shedding light on the changes in TEE design.

5.3 Case Study: Evolution of Protection Mechanisms in AMD SEV

In this section, we perform a detailed analysis of attacks on SEV resulting from vulnerable system design and discuss the underlying causes of these attacks.

5.3.1 *Context Switch.*

In AMD SEV and SEV-ES, they adopt the unprotected mode and RTPM-guarded mode respectively during context switching. This means that during the context switch process, all/some resources are managed by the untrusted hypervisor. This has led to some attacks aimed at the context switch phase, including *Unencrypted register attacks*, *TLB poisoning attack*, and *Unauthenticated encryption attack*.

Unencrypted Register Attacks. Without SEV-ES and SEV-SNP extensions, the VMEXIT operation in the baseline SEV [44] (unprotected mode) acts very similar to traditional context switches in virtualization. The hardware saves the VM's register values directly to the VM control block (VMCB) area and restores the host's register values from the host save area. Additionally, during the VMEXIT and VMRUN operations, CPU updates the "ASID" register, which stores an ID to differentiate between the host and different TEE instances.

The unencrypted VMCB and unprotected mode lead to various attacks [32, 78, 93] that breach both the confidentiality and integrity of the SEV-protected VM. For example, the SEVerESt attack [93] showed that by continuously monitoring register values stored in the VMCB area after the VMEXIT operation, an attacker can infer the instructions being executed or fingerprint applications inside the SEV VM. Similarly, Hetzelt *et al.* [32] showed that an attacker could modify RIP register after VMEXIT to perform Return-oriented programming (ROP) attacks. AMD fixed the register leakage vulnerability in SEV-ES and SEV-SNP by leveraging RTPM to protect VM's register values during context switch via encryption. Specifically, CPU microcodes in SEV-ES and SEV-SNP now encrypt VM's register values and store the ciphertext in an VM Save Area (VMSA).

TLB Poisoning Attacks. In SEV and SEV-ES, RTPM does not actively perform TLB flush during VMEXIT or VMRUN operations. This performance optimization aligns with the traditional KVM design without TEE, where frequent TLB flushes can be avoided by assigning a unique Address Space ID (ASID) to VM's vCPUs, and such ASID tag prohibits illegal access to dirty TLB entries.

However, for SEV-protected VMs, ASID needs to be consistent for all vCPUs from the same VM as ASID is also identified by the hardware for choosing the memory encryption key. Thus, SEV leverages the untrusted host to enforce a TLB flush in some special cases, such as context switching between vCPUs from the same VM on the same logical CPU core. TLB Poisoning attack explored this hypervisor-controlled TLB flush and showed that attackers could breach TLB isolation between two vCPUs from a victim VM. Specifically, the update of the microarchitectural states for this attack pattern is explained in Figure 6. We refer to the two vCPUs as "victim vCPU" and "attack vCPU". In the example presented, the attack vCPU aims to directly utilize the TLB entry residues from the victim vCPU. As shown in Figure 6, during the victim vCPU's VMEXIT, RTPM only clears the registers but does not flush TLB. Later, during the attacker vCPU's VMRUN, the attacker's registers are loaded. When the attacker subsequently performs a read operation with a program counter value (stored in RIP register) that is used by the victim vCPU before and remains cached in the TLB, the TLB entry from the victim will be then directly inherited and used due to them sharing the identical ASID. This pattern of operation sequence, in turn, compromises the integrity of the victim VM.

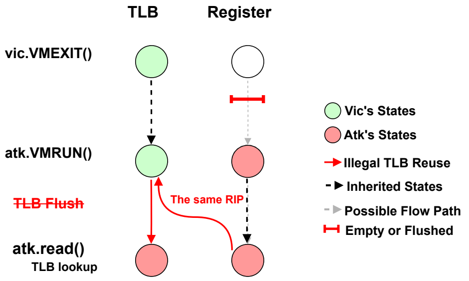


Figure 6: Sample diagram for TLB Poisoning attacks.

AMD SEV-SNP mitigates the TLB Poisoning attacks by using the RTPM-only mode during context switch. Specifically, the CPU microcode now performs TLB flush checks and enforces TLB flush when necessary by recording vCPU states inside a hardware-protected region [4]. By eliminating the untrusted hypervisor's control over TLB resources, such attacks can be mitigated.

Unauthenticated Encryption Attacks. Unauthenticated Encryption Attacks [53] are carried out by tampering with the ASID during context switch, leading to the use of incorrect memory encryption keys. In AMD SEV and SEV-ES, an attacker can modify the ASID stored in the VMCB after a VMEXIT, causing the subsequent VMRUN to use the wrong key to decrypt accessed memory. Starting with AMD SEV-SNP with RTPM-only mode, the hardware checks the ownership of all accessed pages during VMRUN, effectively mitigating this type of attack.

5.3.2 *Instruction Emulation.*

In all three versions of SEV, the unprotected mode is employed for instruction emulation. This implies that TEE instances directly rely on the emulation results supplied by the hypervisor. Consequently, the hypervisor can exert control over a TEE instance's operation by supplying inaccurate register values. To mitigate this risk, the TEE instance might implement internal software checks to filter out potentially erroneous emulations. However, such defenses are neither enforced nor comprehensive.

Wrong CPUID Information Attacks. One example is wrong CPUID value [3]. Specifically, the hypervisor can provide an incorrect size for the x86 Extended Save Area to the victim VM, causing the VM to allocate a smaller memory region than expected. Subsequently, a potential buffer overflow can be triggered when the VM instance later executes XSAVE instructions. To address this issue, SEV-SNP adds an additional CPUID filtering feature through the AMD Secure Processor (AMD-SP). AMD-SP can roughly verify the CPUID result to ensure that it falls within a reasonable range.

5.3.3 *Memory Allocation.*

In both SEV and SEV-ES, a weak form of RTPM-guarded mode is employed to safeguard memory allocation. Specifically, memory allocation is protected using memory encryption keys provided by RTPM. For instance, if the untrusted hypervisor maps a memory page from TEE instance A to an existing memory page of TEE instance B, instance A can gain access to the memory page of instance B. However, since that memory page is encrypted with a key unique to instance B, instance A will not be able to retrieve the protected plaintext. Nevertheless, such weak RTPM-guarded mode overlooks the mapping information of memory allocation within a TEE instance, leading to a series of attacks stemming from the unprotected nested page table (NPT).

Unprotected Nested Page Table Attacks. In AMD SEV and SEV-ES, NPT is maintained by the untrusted hypervisor. The hypervisor can remap a victim VM's guest physical address to another system physical address also owned by the same VM to change its control flow or break its data confidentiality. Hetzelt *et al.* [32] first discussed the possibility of changing the VM's execution flow by redirecting the address translation. The similar method can also be applied to data pages [65, 66, 67]. In the SEVered attack [66], the attacker targets some network applications (*e.g.*, webserver) and remaps the guest physical address from a system physical address that is supposed to contain network data to arbitrary system physical addresses that contain the victim VM's data.

6 DISCUSSION

6.1 Other Attacks

Besides the vulnerabilities stemming from TEE design flaws, there are also various other types of attack against TEE (Table 2). While some attacks are excluded from the TEE's threat model (*e.g.*, side-channel attacks), current research indicates that these attacks have emerged as genuine threats against TEE.

Side-channel attacks. Although side channel attacks have already been shown to be a threat in the cloud environment [59, 100], the strong assumption of a privileged attacker makes side-channel attacks even more powerful in TEE environment for two reasons. Firstly, the attacker has access to privileged interfaces, such as the performance counter [30] and power consumption analyzer [57], which can be used to gain more side channels for inference. Secondly, the attacker can gather more fine-grained information by intentionally controlling global resources [12, 24, 62, 77, 84], such as setting CPU affinity or using controlled-channel attacks.

Consequently, some traditional side-channel attacks bring additional threats against TEE designs. For example, CacheQuote [24] challenged the security of remote attestation procedure used by Intel SGX. Since the attestation supervisors (quoting enclaves) also share the CPU resources with other untrusted threads, CacheQuote showed it could utilize interrupt-controlled L1 cache side-channel to successfully leak long-term secret used by the Extended Privacy ID (EPID) protocol in the remote attestation procedure. Additionally, some side-channel attacks utilize new side-channel information to infer secret from within the TEE. One particular type of side-channel attacks is ciphertext side-channel attacks against AMD SEV, which monitors the ciphertext after memory encryption to infer secret [52, 55]. Furthermore, performance counters, power consumption interfaces, or CPU frequency could also be abused to monitor TEE instances to leak sensitive data [30, 52, 57].

Controlled-channel attacks. Controlled-channel attacks are attacks where the privileged attacker can control some resources used by TEE instances, such as page tables or CPU scheduling, to intentionally pause their execution, which provides additional information and proper time windows to conduct other attacks (*e.g.*, side-channel attacks). Controlled-channel attacks are usually due to choosing RTPM-guarded or unprotected mode.

Interrupt-based controlled-channel attacks. Most TEE designs adopt a non-protection mode in CPU scheduling (§4.3), leaving attackers the ability to force a TEE instance to exit and conduct fine-grained attacks. For example, SGX-STEP [83] first showed that an attacker could use APIC timer interrupts to single-step SGX enclaves' execution in instruction-level granularity. Similar attacks were also studied in the AMD platform to monitor SEV VM's states. SEV-STEP [95] and CipherLeaks attack [55] used the timer interrupt to step SEV VM's execution inside an instruction page. Such time interrupt-based controlled channel can be used combining with other vulnerabilities (*e.g.*, cache side-channel) to gain fine-grained attack windows. To address these controlled-channel attacks, recent academic research has been focusing on reducing the impact of untrusted OS through software-hardware co-design [21, 70].

Page table-based controlled-channel attacks. As introduced in §4.4.3, TEE designs that do not adopt the RTPM-only mode in page table maintenance, such as Intel SGX and AMD SEV, will suffer from page table-based controlled-channel attacks [32, 54, 65, 66, 67, 79, 85, 89, 93, 98]. By manipulating the page table entry (PTE) in the page table (*e.g.*, clear the "present" bit or "dirty" bit), the attacker can intentionally raise page faults or observe the time points when the TEE instance tries to access certain memory pages. These exceptions or observations immediately abort the execution of the TEE instance to help the attacker intercept the TEE instance at desired execution points or infer its behaviors. For example, by collecting the trace of memory access at the page level, pigeonhole [79] showed it could steal secret keys used by the SGX enclave when cryptography implementations have a secret-dependent trace of page access.

Speculative Execution. Speculative execution attacks are highly affected by CPU designs on different platforms. Many CPU vendors opt to implement permission-related check in parallel with speculatively execution for performance reason, which was first exploited on Intel CPU by Meltdown [58] and Spectre [46] attacks to bypass software-based permission isolation and use side-channel to leak information. Subsequently, numerous research works have delved into speculative execution-related attacks across various microarchitectures, including L1 data cache [86, 88], Branch Target Buffer (BTB) [3, 16], Return Stack Buffer (RSB) [47], Line Fill Buffer (LFB) [35, 76, 82, 86, 87, 88], Single instruction multiple data (SIMD) buffer [63], *etc.*. Due to the strong threat model of TEE, TEE designs on different platforms have become a prime target for many speculative attacks. It's worth noting that speculative execution is more of a general issue related to isolation between security domains, rather than solely a design issue specific to TEE.

Other Microarchitectural Flaws. In addition to speculative execution attacks, there are other attacks stemming from microarchitecture flaws. These attacks exploit hardware issues within SoC, leading to TEE data to be accessed or manipulated. These attacks violate the TEE's threat model, where the SoC itself is considered trusted. Thus, they are often unrelated to high-level TEE designs, making them more challenging to detect. For instance, in the AMD

Zen1 architecture, it was discovered that the memory encryption engine used insecure encryption modes and non-randomized tweak functions [27, 94]. This allowed attackers to conduct ciphertext replay attacks. Another type of microarchitectural flaw is illegal data forwarding, where attackers can gain unauthorized access to or infer stale data remaining in TEE instances using side channels or undefined registers. These attacks often result from incomplete CPU designs that allow certain instructions [99] or hidden buffers [10, 37] to bypass the isolation between the TEE and the untrusted world.

Hardware Flaws. Hardware flaws could also be leveraged to compromise TEEs. For example, the processor might behave abnormally when voltage drops. Hence, when the adversary could manipulate the processor's voltage via physical access or software interfaces when running TEE instance, the security checks might be bypassed, resulting in breach of confidentiality and even integrity [17, 45, 69, 73]. Another example of hardware flaws is Rowhammer attacks, which are based on a known hardware flaw in DRAM that triggers random bit flips, which could be abused in TEE configurations to lock down the processor [40].

6.2 Limitations of TRAF

TRAF focuses on conducting a thorough and systematic disassembly of TEE, analyzing their design choices and implications. However, this high-level disassembly primarily serves to highlight the considerations and trends of TEE designs, but it is insufficient to assess the security of detailed implementations of a TEE design. The main reason why TRAF does not choose to include a security analysis from a microarchitectural perspective is that the substantial divergence in the specific implementations of TEE designs increases the challenge of performing a systematization of every implementation detail. Enumerating the implementation details for each task can easily confuse new researchers and deviate from the intended purpose of a SoK paper. In contrast, due to the unique hierarchical model introduced by TEE (where a TEE instance is managed by the hypervisor but must prevent information leakage), determining how to securely manage resources under this distinctive threat model presents a new challenge for server OSes with TEE requirement. With this SoK paper, we aim to reveal the appropriate design choices in different runtime tasks with TEE context.

7 RELATED WORK

Some existing work has also made significant efforts to summarize or compare TEE designs from various perspectives, such as systematizing protection mechanisms, performance comparisons, and conducting in-depth examinations of specific TEE designs.

Existing SoK works about TEE. Schneider *et al.* [75] also investigated how hardware-based TEE designs offer security. Instead of focusing on high-level design choices, such as how management tasks are coordinated between the untrusted OS and RTPM, they primarily delve into the implementation details of TEE designs. They categorize and point out that TEE designs typically provide data isolation through spatial, temporal, or hybrid approaches. Meanwhile, they focus on a broader range of TEE designs, including not only server-level TEEs but also embedded TEE designs. However, the multitude of TEE implementation details and diverse working scenarios may lead to a scattered classification, making it challenging for new researchers to grasp the focal points of TEE design. In contrast, our SoK begins with the challenges of resource management in TEE, proposing a structured framework similar to knowledge classification in OS textbooks. This framework can then be used to analyze TEE considerations of efficiency, performance, and security in various runtime tasks and their potential security threats.

Zhao *et al.* [101] provide detailed explanations of both software-based and hardware-based TEEs. However, the distinction between software and hardware-based TEEs leads to failures in establishing a structured analytical framework, as we do in this paper. Paju *et al.* [71] conduct a study involving more than 200 trusted applications built upon TEEs from both academia and industry side. They cover various application scenarios and the design challenges encountered when developing trusted applications on different TEE platforms, including TCB size, performance, and usability. Mofrad *et al.* [61] and Akram *et al.* [1, 2] provide an overview and SoK of TEE performance under various working scenarios, exploring different performance bottlenecks of TEEs.

Studies of a specific TEE design. Some works focused on the explanation of a specific TEE design [15, 22]. Costan *et al.* [22] serve as a great educational document of Intel SGX and inspire many following-up works about enclave-based TEEs. Cerdeira *et al.* [15] study ARM TrustZone and existing secure systems built upon TrustZone and Cortex-A processors. Lu *et al.* [60] and Dessouky *et al.* [26] summarize academic TEE implementations on RISC-V.

Other summarizing works on TEEs. Munoz *et al.* [68] provide a survey of attacks against TEEs, including software attacks, architectural attacks, side-channel attacks, and microarchitectural attacks. They also discuss the corresponding countermeasures. Sabt *et al.* [74] define TEE and its general security properties. Demigha *et al.* [25] enumerates four famous TEE designs, analyzes their supported features, and discusses applicable working scenarios. Jauernig *et al.* [41] delve into security properties, applications, and future directions within the confidential cloud industry.

8 CONCLUSION

In this paper, we systematize the design choices and security implications of hardware-based server-level TEEs. We propose the TRAF framework as a means to deconstruct TEE designs, enabling a deeper understanding of existing TEEs based on their overarching design choices. Through our analysis of existing TEE designs, we show a clear trend of adopting similar design choices by most TEEs. In addition, by analyzing existing attacks on TEEs, we examined the potential impacts of various TEE design choices and delved into the reasons behind the emergence of these attacks.

ACKNOWLEDGEMENTS

Yinqian Zhang is supported in part by Key Special Project of the National Key Research and Development Program No. 2023YFB4503902, National Natural Science Foundation of China No. 62361166633 and Shenzhen Science and Technology Program No. JSGG2022083109560 3007. This work was also funded in part by the Air Force Office of Scientific Research (AFOSR) under grants FA9550-22-1-0511.

A A LIST OF ATTACKS AGAINST TEES

	Attack Surface	Vendor	TEE Type	Authors	Physical	Confidentiality	Integrity
Vulnerable Design	Unprotected Regs	AMD SEV	VM	Hetzelt *et al.* [32]	✗	✓	✗
	Unprotected Regs	AMD SEV	VM	Werner *et al.* [93]	✗	✓	✓
	Unprotected NPT	SEV/SEV-ES	VM	Morbitzer *et al.* [65, 66]	✗	✓	✗
	Unprotected NPT	SEV/SEV-ES	VM	Li *et al.* [53]	✗	✓	✗
	Unprotected NPT	SEV/SEV-ES	VM	Morbitzer *et al.* [67]	✗	✓	✓
	Unauthenticated Encryption	SEV/SEV-ES	VM	Li *et al.* [53]	✗	✓	✗
	TLB Misuse	SEV/SEV-ES	VM	Li *et al.* [56]	✗	✓	✓
	Debug Registers	SEV/ES/SNP	VM	AMD *et al.* [3]	✗	✓	✓
	Untrusted CPUID	SEV/ES/SNP	VM	AMD *et al.* [3]	✗	✓	✗
	Remote attestation & firmware	AMD SEV	VM	Buhren *et al.* [14]	✗	✓	✗
Controlled Side-Channel	Page Table	Intel SGX	Enclave	Xu *et al.* [98]	✗	✓	✗
	Page Table	Intel SGX	Enclave	Shinde *et al.* [79]	✗	✓	✗
	Page Table	Intel SGX	Enclave	Wang *et al.* [89]	✗	✓	✗
	Page Table	Intel SGX	Enclave	Van Bulck *et al.* [85]	✗	✓	✗
	Page Table	SEV/SEV-ES	VM	Werner *et al.* [93]	✗	✓	✗
	Timer Interrupt	Intel SGX	Enclave	Van Bulck *et al.* [83, 84]	✗	✓	✗
	Timer Interrupt	AMD SEV	VM	Li *et al.* [55]	✗	✓	✗
	Timer Interrupt	AMD SEV	VM	Van Wilke *et al.* [95]	✗	✓	✗
Micro-architectural Side-Channel	L1/ L2 Cache	Intel SGX	Enclave	Götzfried *et al.* [30]	✗	✓	✗
	L1/ L2 Cache	Intel SGX	Enclave	Moghimi *et al.* [62]	✗	✓	✗
	L1/ L2 Cache	Intel SGX	Enclave	Dall *et al.* [24]	✗	✓	✗
	L1/ L2 Cache	Intel SGX	Enclave	Brasser *et al.* [12]	✗	✓	✗
	Last-level Cache	Intel SGX	Enclave	Schwarz *et al.* [77]	✗	✓	✗
	Memory Bus	Intel SGX	Enclave	Lee *et al.* [48]	✓	✓	✗
	Branch Predictor	Intel SGX	Enclave	Lee *et al.* [50]	✗	✓	✗
	Branch Predictor	Intel SGX	Enclave	Evtyushkin *et al.* [28]	✗	✓	✗
	Branch Predictor	Intel SGX	Enclave	Huo *et al.* [34]	✗	✓	✗
	PMC	Intel SGX	Enclave	Götzfried *et al.* [30]	✗	✓	✗
	PMC	AMD SEV	VM	Li *et al.* [52]	✗	✓	✗
	Power-Consumption	Intel SGX	Enclave	Lipp *et al.* [57]	✗	✓	✗
	Power-Consumption	AMD SEV	VM	Wang *et al.* [90]	✗	✓	✗
	Ciphertext	AMD SEV	VM	Li *et al.* [52, 55]	✗	✓	✗
Speculative Execution	L1 Cache	Intel SGX	Enclave	Van Bulck *et al.* [81]	✗	✓	✗
	BTB	Intel SGX	Enclave	Chen *et al.* [16]	✗	✓	✗
	RSB	Intel SGX	Enclave	Koruyeh *et al.* [47]	✗	✓	✗
	LFB	Intel SGX	Enclave	Schwarz *et al.* [76]	✗	✓	✗
	LFB	Intel SGX	Enclave	Van Schaik *et al.* [86, 87, 88]	✗	✓	✗
	LFB	Intel SGX	Enclave	Intel [35]	✗	✓	✗
	LFB	Intel SGX	Enclave	Van Bulck *et al.* [82]	✗	✓	✗
	SIMD	Intel SGX	Enclave	Moghimi [63]	✗	✓	✗
μarchitecture Flaw	Weak Encryption	SEV/SEV-ES	VM	Du *et al.* [27]	✗	✓	✗
	Weak Encryption	SEV/SEV-ES	VM	Wilke *et al.* [94]	✗	✓	✓
	Weak Encryption & I/O	SEV/SEV-ES	VM	Li *et al.* [54]	✗	✓	✗
	INVD Instruction	SEV-ES/SNP	VM	Zhang *et al.* [99]	✗	✓	✓
	APIC	Intel SGX	Enclave	Borrello *et al.* [10]	✗	✓	✗
	MMIO stale data	Intel SGX	Enclave	Intel [37]	✗	✓	✗
Hardware Flaw	Voltage glitching	Intel SGX	Enclave	Qiu *et al.* [73]	✗	✓	✓
	Voltage glitching	Intel SGX	Enclave	Murdock *et al.* [69]	✗	✓	✓
	Voltage glitching	Intel SGX	Enclave	Kenjar *et al.* [45]	✗	✓	✓
	Voltage glitching	Intel SGX	Enclave	Chen *et al.* [17]	✓	✓	✓
	Voltage glitching	AMD SEV	VM	Buhren *et al.* [13]	✓	✓	✓
	Rowhammer	Intel SGX	Enclave	Jang *et al.* [40]	✗	✓	✗

Table 2: Existing attacks. Column "Physical" indicates whether the attacks require physical accesses. Column "Confidentiality" and "Integrity" indicate the security properties the attacks aim to compromise.

REFERENCES

[1] Ayaz Akram, Venkatesh Akella, Sean Peisert, and Jason Lowe-Power. 2022. SoK: Limitations of Confidential Computing via TEEs for High-Performance Compute Systems. In *2022 IEEE International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 121–132.

[2] Ayaz Akram, Anna Giannakou, Venkatesh Akella, Jason Lowe-Power, and Sean Peisert. 2021. Performance analysis of scientific computing workloads on general purpose TEEs. In *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 1066–1076.

[3] AMD. 2020. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. *White paper* (2020).

[4] AMD. 2021. Security Bulletin fo TLB Poisoning Attacks on AMD Secure Encrypted Virtualization (SEV). https://www.amd.com/en/corporate/product-security/bulletin/amd-sb-1023.

[5] ARM. 2021. ARM CCA Security Model 1.0.

[6] ARM. 2022. Trustzone For Cortex-A. https://www.arm.com/technologies/trustzone-for-cortex-a.

[7] Remzi H Arpaci-Dusseau and Andrea C Arpaci-Dusseau. 2018. *Operating systems: Three easy pieces.* Arpaci-Dusseau Books LLC Boston.

[8] AWS. 2022. AWS Nitro Enclaves. https://aws.amazon.com/ec2/nitro/nitro-enclaves/.

[9] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2021. CURE: A Security Architecture with CUstomizable and Resilient Enclaves. In *30th USENIX Security Symposium (USENIX Security 21)*.

[10] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. 2022. {ÆPIC} Leak: Architecturally Leaking Uninitialized Data from the Microarchitecture. In *31st USENIX Security Symposium (USENIX Security 22)*. 3917–3934.

[11] Justinien Bouron, Sebastien Chevalley, Baptiste Lepers, Willy Zwaenepoel, Redha Gouicem, Julia Lawall, Gilles Muller, and

Julien Sopena. 2018. The Battle of the Schedulers:FreeBSD ULE vs. Linux CFS. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 85–96.

[12] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure:SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*.

[13] Robert Buhren, Hans-Niklas Jacob, Thilo Krachenfels, and Jean-Pierre Seifert. 2021. One Glitch to Rule Them All: Fault Injection Attacks Against AMD's Secure Encrypted Virtualization. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 2875–2889.

[14] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. 2019. Insecure until proven updated: analyzing AMD SEV's remote attestation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1087–1099.

[15] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. 2020. Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1416–1432.

[16] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2019. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 142–157.

[17] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D Garcia. 2021. VoltPillager: Hardware-based fault injection attacks against Intel SGX Enclaves using the SVID voltage scaling interface. In *30th USENIX Security Symposium (USENIX Security 21)*. 699–716.

[18] Tobias Cloosters, Michael Rodler, and Lucas Davi. 2020. TeeRex: Discovery and Exploitation of Memory Corruption Vulnerabilities in SGX Enclaves. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 841–858. https://www.usenix.org/conference/usenixsecurity20/presentation/cloosters

[19] Confidential Computing Consortium. 2020. Confidential Computing: Hardware-Based Trusted Execution for Applications and Data. https://confidentialcomputing.io/wp-content/uploads/sites/85/2020/06/ConfidentialComputing_OSSNA2020.pdf.

[20] Confidential Computing Consortium. 2022. Confidential Computing Consortium Members. https://confidentialcomputing.io/members/.

[21] Scott Constable, Jo Van Bulck, Xiang Cheng, Yuan Xiao, Cedric Xing, Ilya Alexandrovich, Taesoo Kim, Frank Piessens, Mona Vij, and Mark Silberstein. 2023. AEX-Notify: Thwarting Precise Single-Stepping Attacks through Interrupt Awareness for Intel SGX Enclaves. In *32nd USENIX Security Symposium (USENIX Security 23)*. 4051–4068.

[22] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* 2016, 86 (2016), 1–118.

[23] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *25th USENIX Security Symposium (USENIX Security 16)*. 857–874.

[24] Fergus Dall, Gabrielle De Micheli, Thomas Eisenbarth, Daniel Genkin, Nadia Heninger, Ahmad Moghimi, and Yuval Yarom. 2018. Cachequote: Efficiently recovering long-term secrets of SGX EPID via cache attacks. (2018).

[25] Oualid Demigha and Ramzi Larguet. 2021. Hardware-based solutions for trusted cloud computing. *Computers & Security* 103 (2021), 102117.

[26] Ghada Dessouky, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2020. Enclave computing on RISC-V: A brighter future for security?. In *Workshop on Secure RISC-V Architecture Design (SECRISC-V'20)*.

[27] Zhao-Hui Du, Zhiwei Ying, Zhenke Ma, Yufei Mai, Phoebe Wang, Jesse Liu, and Jesse Fang. 2017. Secure Encrypted Virtualization is Unsecure. *arXiv preprint arXiv:1712.05090* (2017).

[28] Dmitry Evtyushkin, Ryan Riley, Nael CSE Abu-Ghazaleh, ECE, and Dmitry Ponomarev. 2018. Branchscope: A new side-channel attack on directional branch predictor. *ACM SIGPLAN Notices* 53, 2 (2018), 693–707.

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

[30] Johannes Götzfried, Moritz Eckert, Sebastian Schinzel, and Tilo Müller. 2017. Cache attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security*. 1–6.

[31] Michael Gruhn and Tilo Müller. 2013. On the practicability of cold boot attacks. In *2013 International Conference on Availability, Reliability and Security*. IEEE, 390–397.

[32] Felicitas Hetzelt and Robert Buhren. 2017. Security analysis of encrypted virtual machines. In *ACM SIGPLAN Notices*. ACM.

[33] Guerney DH Hunt, Ramachandra Pai, Michael V Le, Hani Jamjoom, Sukadev Bhattiprolu, Rick Boivie, Laurent Dufour, Brad Frey, Mohit Kapur, Kenneth A Goldman, et al. 2021. Confidential computing for OpenPOWER. In *Proceedings of the Sixteenth European Conference on Computer Systems*. 294–310.

[34] Tianlin Huo, Xiaoni Meng, Wenhao Wang, Chunliang Hao, Pei Zhao, Jian Zhai, and Mingshu Li. 2020. Bluethunder: A 2-level directional predictor based side-channel attack against SGX. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2020), 321–347.

[35] Intel. 2019. Intel Transactional Synchronization Extensions (Intel TSX) Asynchronous Abort. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/intel-tsx-asynchronous-abort.html.

[36] Intel. 2020. Intel Trust Domain Extensions Whitepaper. https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-final9-17.pdf.

[37] Intel. 2022. Processor MMIO Stale Data Vulnerabilities. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/technical-documentation/processor-mmio-stale-data-vulnerabilities.html.

[38] Intel. 2022. Software Enabling for Intel® TDX in Support of TEE-IO. *White paper* (2022).

[39] Intel. 2022. What Are PCIe 4.0 and 5.0? https://www.intel.com/content/www/us/en/gaming/resources/what-is-pcie-4-and-why-does-it-matter.html.

[40] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking down the processor via Rowhammer attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 1–6.

[41] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2020. Trusted execution environments: properties, applications, and challenges. *IEEE Security & Privacy* 18, 2 (2020), 56–60.

[42] Yuekai Jia, Shuang Liu, Wenhao Wang, Yu Chen, Zhengde Zhai, Shoumeng Yan, and Zhengyu He. 2022. HyperEnclave: An Open and Cross-platform Trusted Execution Environment. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*. 437–454.

[43] David Kaplan. 2017. Protecting VM register state with SEV-ES. *White paper* (2017).

[44] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).

[45] Zijo Kenjar, Tommaso Frassetto, David Gens, Michael Franz, and Ahmad-Reza Sadeghi. 2020. V0LTpwn: Attacking x86 Processor Integrity from Software. In *29th USENIX Security Symposium (USENIX Security 20)*. 1445–1461.

[46] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–19.

[47] Esmaeil Mohammadian Koruyeh, Khaled N Khasawneh, Chengyu Song, and Nael Abu-Ghazaleh. 2018. Spectre returns! speculation attacks using the return stack buffer. In *12th USENIX Workshop on Offensive Technologies (WOOT 18)*.

[48] Dayeol Lee, Dongha Jung, Ian T Fang, Chia-Che Tsai, and Raluca Ada Popa. 2020. An Off-Chip Attack on Hardware Enclaves via the Memory Bus. In *29th USENIX Security Symposium (USENIX Security 20)*.

[49] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An open framework for architecting trusted execution environments. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[50] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside {SGX} enclaves with branch shadowing. In *26th USENIX Security Symposium (USENIX Security 17)*. 557–574.

[51] Dingji Li, Zeyu Mi, Yubin Xia, Binyu Zang, Haibo Chen, and Haibing Guan. 2021. TwinVisor: Hardware-isolated Confidential Virtual Machines for ARM. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. 638–654.

[52] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. 2022. A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, 1541–1541.

[53] Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. 2020. CROSSLINE: Breaking "Security-by-Crash" based Memory Isolation in AMD SEV. *arXiv preprint arXiv:2008.00146* (2020).

[54] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. 2019. Exploiting Unprotected I/O Operations in AMD's Secure Encrypted Virtualization. In *28th USENIX Security Symposium*. 1257–1272.

[55] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. 2021. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *30th USENIX Security Symposium (USENIX Security 21)*. 717–732.

[56] Mengyuan Li, Yinqian Zhang, Huibo Wang, Kang Li, and Yueqiang Cheng. 2021. TLB Poisoning Attacks on AMD Secure Encrypted Virtualization. In *Annual Computer Security Applications Conference*. 609–619.

[57] Moritz Lipp, Andreas Kogler, David Oswald, Michael Schwarz, Catherine Easdon, Claudio Canella, and Daniel Gruss. 2021. PLATYPUS: software-based power side-channel attacks on x86. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 355–371.

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

[59] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*. IEEE, 605–622.

[60] Tao Lu. 2021. A survey on risc-v security: Hardware and architecture. *arXiv preprint arXiv:2107.04175* (2021).

[61] Saeid Mofrad, Fengwei Zhang, Shiyong Lu, and Weidong Shi. 2018. A comparison study of Intel SGX and AMD memory encryption technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*. 1–8.

[62] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. Cachezoom: How SGX amplifies the power of cache attacks. In *International Conference on Cryptographic Hardware and Embedded Systems*. Springer, 69–90.

[63] Daniel Moghimi. 2023. Downfall: Exploiting Speculative Data Gathering. In *32nd USENIX Security Symposium (USENIX Security 23)*. 7179–7193.

[64] Hyungon Moon, Hojoon Lee, Ingoo Heo, Kihwan Kim, Yunheung Paek, and Brent Byunghoon Kang. 2015. Detecting and preventing kernel rootkit attacks with bus snooping. *IEEE Transactions on Dependable and Secure Computing* 14, 2 (2015), 145–157.

[65] Mathias Morbitzer, Manuel Huber, and Julian Horsch. 2019. Extracting Secrets from Encrypted Virtual Machines. In *9th ACM Conference on Data and Application Security and Privacy*. ACM.

[66] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. 2018. SEVered: Subverting AMD's Virtual Machine

Encryption. In *11th European Workshop on Systems Security*. ACM.

[67] Mathias Morbitzer, Sergej Proskurin, Martin Radev, Marko Dorfhuber, and Erick Quintanar Salas. 2021. Severity: Code injection attacks against encrypted virtual machines. In *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE, 444–455.

[68] Antonio Muñoz, Ruben Ríos, Rodrigo Román, and Javier López. 2023. A survey on the (in) security of trusted execution environments. *Computers & Security* 129 (2023), 103180.

[69] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based fault injection attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1466–1482.

[70] Meni Orenbach, Andrew Baumann, and Mark Silberstein. 2020. Autarky: Closing controlled channels with self-paging enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems*. 1–16.

[71] Arttu Paju, Muhammad Owais Javed, Juha Nurmi, Juha Savimäki, Brian McGillion, and Billy Bob Brumley. 2023. SoK: A Systematic Review of TEE Usage for Developing Trusted Applications. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*. 1–15.

[72] Chad Perrin. 2008. The CIA triad. *Dostopno na* (2008).

[73] Pengfei Qiu, Dongsheng Wang, Yongqiang Lyu, and Gang Qu. 2019. VoltJockey: Breaking SGX by software-controlled voltage-induced hardware faults. In *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 1–6.

[74] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 57–64.

[75] Moritz Schneider, Ramya Jayaram Masti, Shweta Shinde, Srdjan Capkun, and Ronald Perez. 2022. SoK: Hardware-supported Trusted Execution Environments. *arXiv preprint arXiv:2205.12742* (2022).

[76] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-privilege-boundary data sampling. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 753–768.

[77] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware guard extension: Using SGX to conceal cache attacks. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–24.

[78] Hovav Shacham. 2007. The Geometry of Innocent Flesh on the Bone: Return-into-libc Without Function Calls (on the x86). In *14th ACM Conference on Computer and Communications Security*. ACM.

[79] Shweta Shinde, Zheng Leong Chua, Viswesh Narayanan, and Prateek Saxena. 2016. Preventing page faults from telling your secrets. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*. 317–328.

[80] Pramod Subramanyan, Rohit Sinha, Ilia Lebedev, Srinivas Devadas, and Sanjit A Seshia. 2017. A formal foundation for secure remote execution of enclaves. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2435–2450.

[81] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. 991–1008.

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

[83] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2017. SGX-Step: A practical attack framework for precise enclave execution control. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 1–6.

[84] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 178–195.

[85] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. 2017. Telling Your Secrets without Page Faults: Stealthy Page Table-Based Attacks on Enclaved Execution. In *26th USENIX Security Symposium (USENIX Security 17)*. 1041–1056.

[86] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2020. SGAxe: How SGX fails in practice.

[87] Stephan Van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2019. RIDL: Rogue in-flight data load. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 88–105.

[88] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2021. CacheOut: Leaking data on Intel CPUs via cache evictions. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 339–354.

[89] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. 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*. 2421–2434.

[90] Wubing Wang, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. 2023. PwrLeak: Exploiting Power Reporting Interface for Side-Channel Attacks on AMD SEV. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 46–66.

[91] Nico Weichbrodt, Anil Kurmus, Peter Pietzuch, and Rüdiger Kapitza. 2016. AsyncShock: Exploiting synchronisation bugs in Intel SGX enclaves. In *European Symposium on Research in Computer Security*. Springer, 440–457.

[92] Ofir Weisse, Jo Van Bulck, Marina Minkin, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Raoul Strackx,

Thomas F Wenisch, and Yuval Yarom. 2018. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution. (2018).

[93] Jan Werner, Joshua Mason, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. 2019. The SEVerESt Of Them All: Inference Attacks Against Secure Virtual Enclaves. In *ACM Asia Conference on Computer and Communications Security*. ACM, 73–85.

[94] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. 2020. SEVurity: No Security Without Integrity: Breaking Integrity-Free Memory Encryption with Minimal Assumptions. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1483–1496.

[95] Luca Wilke, Jan Wichelmann, Anja Rabich, and Thomas Eisenbarth. 2023. SEV-Step: A Single-Stepping Framework for AMD-SEV. *arXiv preprint arXiv:2307.14757* (2023).

[96] L. Wilke, J. Wichelmann, F. Sieck, and T. Eisenbarth. 2021. undeSErVed trust: Exploiting Permutation-Agnostic Remote Attestation. In *2021 IEEE Security and Privacy Workshops (SPW)*. IEEE Computer Society, Los Alamitos, CA, USA, 456–466. https://doi.org/10.1109/SPW53761.2021.00064

[97] Yubin Xia, Zhichao Hua, Yang Yu, Jinyu Gu, Haibo Chen, Binyu Zang, and Haibing Guan. 2021. Colony: A privileged trusted execution environment with extensibility. *IEEE Trans. Comput.* 71, 2 (2021), 479–492.

[98] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 640–656.

[99] Ruiyi Zhang, CISPA Helmholtz Center, Lukas Gerlach, Daniel Weber, Lorenz Hetterich, Youheng Lü, Andreas Kogler, and Michael Schwarz. 2024. CacheWarp: Software-based Fault Injection using Selective State Reset. (2024).

[100] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM SIGSAC conference on Computer and Communications Security*. 305–316.

[101] Lianying Zhao, He Shuang, Shengjie Xu, Wei Huang, Rongzhen Cui, Pushkar Bettadpur, and David Lie. 2019. Sok: Hardware security support for trustworthy execution. *arXiv preprint arXiv:1910.04957* (2019).

[102] Shixuan Zhao, Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. 2022. vSGX: Virtualizing SGX Enclaves on AMD SEV. (2022).