SoK: A Comparison Study of Arm TrustZone and CCA

Haoyang Huang[1,2], Fengwei Zhang[2,1,†],
Shoumeng Yan[3], Tao Wei[3], Zhengyu He[3]
[1]*Research Institute of Trustworthy Autonomous Systems, Southern University of Science and Technology*
[2]*Department of Computer Science and Engineering, Southern University of Science and Technology*
[3]*Ant Group*
12232406@mail.sustech.edu.cn, zhangfw@sustech.edu.cn
shoumeng.ysm@antgroup.com, Lenx.wei@antgroup.com, zhengyu.he@antgroup.com

***Abstract*—Arm TrustZone is the most popular hardware-assisted Trusted Execution Environment (TEE) solution on mobile and Internet of Things (IoT) devices. However, this well-established TEE faces significant challenges in deployment to new scenarios, such as cloud computing. In response, Arm introduces the Confidential Compute Architecture (CCA) in Armv9-A as the next-generation Arm TEE solution. Due to different design goals, CCA has significant differences from TrustZone.**

In this paper, we present a comprehensive overview of Arm TrustZone and CCA, analyzing and comparing them across three critical dimensions: flexibility, security, and performance. Furthermore, we summarize the limitations of TrustZone and CCA and discuss potential solutions. By exploring these aspects, our study can provide valuable insights into the strengths and weaknesses of each technology, helping developers and researchers to better understand and select the right technology for their specific needs.

I. Introduction

The Arm architecture has been widely used in mobile devices and Internet of Things (IoT) devices over the past few decades [1], [2]. These devices contain significant user personal data [3], [4], [5], such as passwords and unique biometric details. Since the operating system (OS) is vulnerable, attackers can easily access sensitive data by exploiting it [6], [7]. Consequently, there is a pressing need for robust security mechanisms that can safeguard sensitive data from the system software. To meet this growing demand, hardware-assisted Trusted Execution Environment (TEE) has received much attention as a solution. It offers advantages over previous solutions regarding Trusted Computing Base (TCB) and performance overhead [8], [9].

TrustZone [10] is the representative hardware-assisted TEE solution on the Arm architecture. It aims to protect code and data in TEE from being accessed by software such as OS in the Rich Execution Environment (REE). It guarantees the system's security by partitioning all of SoC's hardware and software resources into the Normal world and the Secure world [11]. Nowadays, TrustZone is widely used in our daily life and plays a critical role in protecting our sensitive data [12], [13], [14]. Many OEMs, such as Samsung and Google, have integrated this technology into their products [15].

[†] The corresponding author.

Over the past few years, cloud vendors have increasingly been providing Arm-based cloud systems due to their advantage in power efficiency over x86 server processors [16], [17]. Arm released the Neoverse series [18], [19] to meet the needs of cloud computing in terms of performance and scalability, However, these CPUs still use TrustZone as their TEE solution. Since TrustZone is initially designed for mobile devices and IoT devices, it cannot meet the requirements of cloud tenants. For example, it is hard for cloud tenants to deploy their code in the Secure world due to the security considerations of vendors [20].

In 2021, Arm introduced Confidential Compute Architecture (CCA) [21] in Armv9-A. The design goals of CCA are to establish a minimal trust chain and ensure attestable trust [22]. These different design goals set CCA apart from TrustZone and motivate us to perform a study focusing on the following research questions:

- **RQ1:** What are the flexibility and security differences between TrustZone and CCA?
- **RQ2:** What is the performance difference between TrustZone and CCA?
- **RQ3:** What limitations of TrustZone does CCA address?

Flexibility indicates how easily and efficiently developers can manage trusted resources. TEEs with high flexibility can reduce development costs and expand the range of application use cases. Security indicates TEEs' ability to protect applications from specific attacks. Due to different security considerations, TEEs may provide different security guarantees. Since flexibility and security are critical factors that affect the adoption of TEEs, we introduce RQ1 (in Sections III and IV) to compare TrustZone and CCA regarding these two aspects.

Performance is another critical factor that affects the adoption of TEEs. We choose a series of benchmarks to evaluate the performance overhead of CCA and TrustZone in different scenarios to answer RQ2 (in Section V). However, there are no commercial devices with CCA enabled. Although evaluating CCA's features on the Arm Fixed Virtual Platform (FVP) [23] using Arm's official firmware is possible, accurately measuring its performance is difficult since the FVP does not offer precise CPU cycle information [24], [25]. To overcome this limitation,

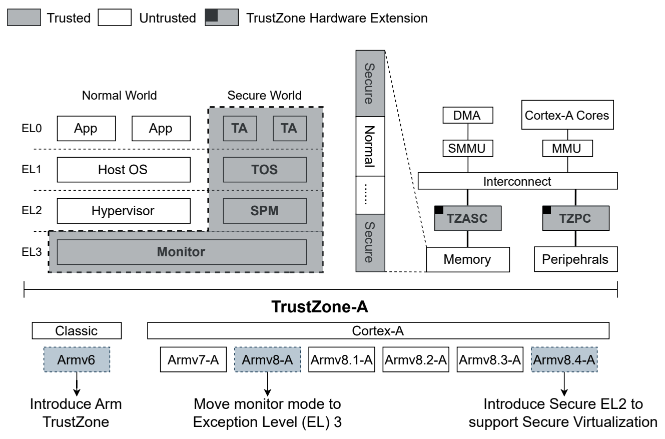

Fig. 1: TrustZone architecture overview.

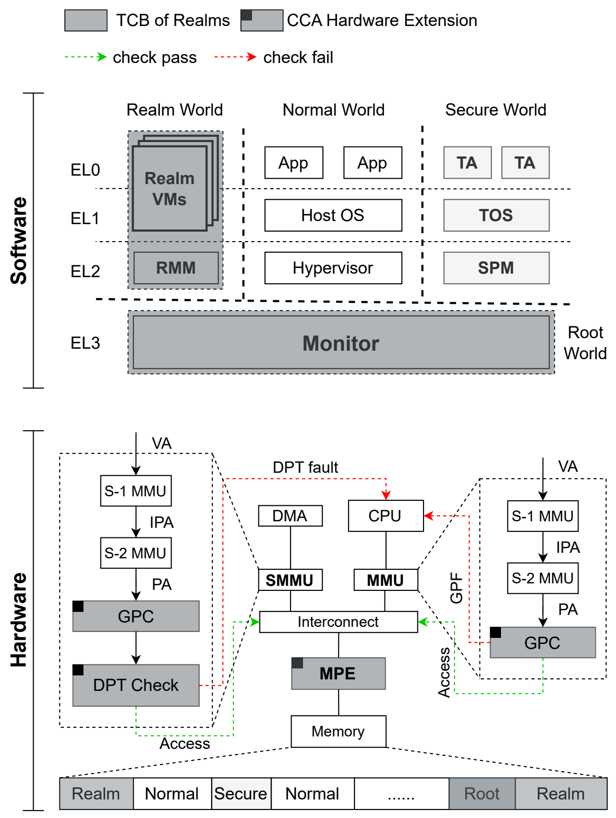

Fig. 2: CCA architecture overview.

we made firmware modifications to evaluate the performance overhead of CCA on a physical development board.

Finally, we present RQ3 (in Section VI) to explore whether TrustZone finally be replaced by CCA. We first summarize the limitations of TrustZone and discuss the approaches that have attempted to address these limitations. We then analyze whether CCA has addressed them entirely and the unique limitations of CCA.

Outline: The rest of this paper is organized as follows. In Section II, we start with an overview of TrustZone and CCA. Then, we explain how to adapt the CCA software stack to a physical development board. Following this, we compare TrustZone and CCA regarding flexibility, security, and performance in Sections III, IV, and V, respectively. We also analyze the limitations of TrustZone and CCA and discuss potential solutions for them in Section VI. Lastly, we introduce some related works in Section VII and provide a conclusion in Section VIII.

II. Overview

A. *TrustZone Overview*

Figure 1 shows the TrustZone architecture overview. It has four Exception Levels (EL0, EL1, EL2, and EL3) and two security states (Secure and Non-secure). Correspondingly, there are two worlds: the Normal world, which hosts complex software like OS in REE, and the Secure world, which hosts trusted applications (TAs) and a lightweight Trusted OS (TOS) in TEE [26]. TrustZone employs two essential components to guarantee isolation between two worlds: the TrustZone Address Space Controller (TZASC) [27] and the TrustZone Protection Controller (TZPC) [28]. They are used for memory protection and peripheral protection, respectively. The TZASC is responsible for partitioning the external memory and configuring the access permission for each memory region. The TZPC can configure the security state of peripherals, thus blocking any unauthorized access attempts. Besides, TrustZone ensures interrupts are routed to their target worlds through the Generic Interrupt Controller (GIC) [29].

B. *CCA Overview*

Figure 2 shows CCA's architecture overview. CCA introduces two additional worlds: the Root world and the Realm world. Correspondingly, there are four security states in CCA, and the system's physical memory is divided into four Physical Address Spaces (PASs). The memory access privilege differs in different security states (Table I).

Roles of each World. The Normal world and the Secure world in CCA are similar to their roles in TrustZone. The difference is that the EL3 in TrustZone belongs to the Secure world, while the EL3 in CCA belongs to the Root world. In the Root world, *Monitor* is responsible for trusted booting and context switch between different worlds. The Realm world is designed to run third-party applications and is isolated from the Normal world and the Secure world. The software running in the Realm world includes Realm virtual machines (VMs) and Realm Management Monitor (RMM). The Realm VM is a confidential VM that the host can dynamically create in the Normal world [21]. RMM provides a set of Realm Management Interfaces (RMIs) to the host that can be used to determine the execution scheduling and memory allocation of Realm VMs. Although the host can manage the Realm VMs, it cannot directly access their content. RMM ensures the isolation between each Realm VM through the two-stage translation, which controls the memory view of each VM.

Granule Protection Check. CCA extends Memory Management Unit (MMU) with Granule Protection Check (GPC) to block invalid access. When translating Virtual Address (VA) to Physical Address (PA), GPC in MMU checks whether the current security state can access the target physical address. When the processor fails in GPC, it will report granule protection check fault (GPF). Besides the access from processors, it is also necessary to check the access to the main memory from Direct Memory Access (DMA) devices. However, since DMA devices can directly access the memory without the intervention of the processor, MMU cannot perform GPC for DMA devices. CCA extends System MMU (SMMU) with GPC to address this issue. SMMU is a hardware component that provides address translation for DMA devices. In this way, invalid DMA access can also be blocked by GPC.

MMU performs GPC based on the Granule Protection Table (GPT), an in-memory structure belonging to the Root world. It has two levels to look up, and the entries in the GPT at different levels have different descriptors (shown in Figure 3). In GPT, Granule Protection Information (GPI) indicates the associated world of granules. GPC can block illegal access by checking whether the current security state has rights to access memory based on the corresponding GPI.

Device Assignment. GPC can only manage memory access permissions in the granularity of worlds. To enable finer control over access, the CCA introduces Device Assignment (DA) [30], which manages devices at the level of Realm VMs. CCA extends SMMU with an additional structure called Device Permission Table (DPT). DPT records the binding relationship between Realm VMs and devices. When a DMA device initiates a DMA access, SMMU can verify whether the target address falls within the memory footprint of the associated Realm VM by consulting the DPT.

Hardware-assisted Memory Encryption. Memory Protection Engine (MPE) is a hardware component that provides memory encryption and integrity in CCA. However, a noteworthy limitation exists: MPE encrypts the entire memory space within the Realm world using a single key. To enhance security, CCA introduces a mechanism known as Memory Encryption Context (MEC), enabling a memory region in the Realm world to be assigned a unique encryption key. In this way, RMM and Realm VMs can be encrypted with a different key. These encryption keys are identified by MECIDs and are securely stored in write-once system registers. In practice, the MECID works with system registers and the page table bits. The page table bits select the encryption key from the corresponding register for a given memory region.

C. CCA Prototype

The CCA prototype consists of three parts: *Monitor*, RMM, and hypervisor. RMM is a decoupled hypervisor that provides sensitive services for Realm VMs, like metadata management and memory mapping. The hypervisor is responsible for scheduling Realm VMs and memory allocation. It makes decisions for Realm VMs through RMIs. *Monitor* is responsible for booting the system and forwarding RMIs. Besides, it also provides services for RMM, such as GPT management [22].

TABLE I: Access permissions in different security states. ”●” denotes access to the target PAS is allowed, and ”–” denotes access is forbidden.

	Root state	Realm state	Secure state	Non-secure state
Root PAS	●	–	–	–
Realm PAS	●	●	–	–
Secure PAS	●	–	●	–
Normal PAS	●	●	●	●

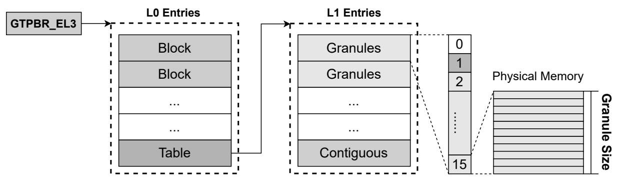

Fig. 3: Overview of Granule Protection Table (GPT). In L0 table, Block descriptor indicates a region (1G to 512G), and Table descriptor indicates the next level Table address. In L1 table, Contiguous descriptor indicates a region (2M to 512M) and Granules descriptor contain GPI values for 16 physical granules. The granule size can be 4KB, 16KB, and 64KB.

CCA Design Analogs. The official CCA software stack has been open-source, including TF-A [31], TF-RMM [32], and patched KVM [33]. They correspond to *Monitor*, RMM, and hypervisor, respectively. While we can assess the features of CCA on FVP, accurately evaluating the performance overhead remains challenging since it cannot provide accurate CPU cycles. Therefore, we modify the CCA software stack and port it to an Armv8 device to obtain a more precise measurement of CCA's performance. Although these modifications do not offer security guarantees due to the lack of CCA-related hardware on an Armv8 device, they can mimic performance costs caused by extensions from CCA.

We observe that RMM and hypervisor do not have any hardware dependencies. Therefore, running RMM and hypervisor on an Armv8 device is similar to running them on the FVP with CCA enabled. However, *Monitor* requires interactions with hardware to support the Realm world and GPT management. Therefore, we need to make two main modifications to *Monitor* to support CCA on an Armv8 device.

First, to support the Realm world, we modify *Monitor* to add another context in the Normal world and move RMM and Realm VMs to this context, which means RMM runs in N-EL2 and Realm VMs run in N-EL0&1. Since no additional registers are added to the Realm world, this new context is the same as the Realm world. The context switch between the Realm world and the Normal world can be simulated by the context switch between these two individual contexts within the Normal world.

Second, to support GPT-related operations, we make two modifications. We can find that not all GPT-related operations need to interact with hardware. GPT is an in-memory structure used for memory partition in CCA. Some GPT-related operations, such as GPT construction, GPT destruction, GPT Entry addition, and GPT Entry deletion, only require modifications to the GPT in memory. For these operations, we only need to add corresponding functions to *Monitor*. As for some operations related to configuring GPT control registers, we simulate them by reading and writing to some unused EL3 registers, such as `ACTLR_EL3`.

Limitations and Bias. We note there are still some performance differences between the Armv8 device and real hardware with RME enabled due to differences in hardware. One such difference is the absence of GPC on the Armv8 device, where its MMU does not perform GPT checks on memory accesses. However, the cost of GPC is expected to be offset by the good caching behavior of future CCA hardware [24]. In addition, the Armv8 device lacks the MPE subsystem. MPE is used in CCA systems to provide memory encryption and integrity, but no specific implementation details of this hardware component is currently available in the manual or FVP. Therefore, we are unable to evaluate the performance overhead caused by MPE.

III. Flexibility Comparison

A. Memory Management

Memory management represents the system's ability to adjust permission settings of memory regions to meet specific requirements. We evaluate this criterion of TrustZone and CCA from five aspects:

- **Dynamic Allocation:** The ability to change the associated world and size of memory regions dynamically.
- **Minimal Granularity:** The smallest configurable size of a memory region.
- **Memory Region Number:** The number of memory regions that can be configured.
- **R/W-separate Configuration:** The ability to configure read and write permissions of memory regions separately.
- **Core-specific Configuration:** The ability to apply different memory partition configurations for different cores.

TrustZone achieves memory partition using TZASC. TZASC does not simply divide memory into the Secure and Non-secure regions. It also allows configuring read and write permissions of the Secure world and Normal world for each region. This feature is beneficial to create regions for specific purposes, such as shared memory between the Secure world and the Normal world. TrustZone allows to configure the size of memory regions during runtime. TZASC is limited in the number of memory areas it can manage [27]. When all the regions are allocated, the only way to expand Secure world memory is to adjust the boundaries of these regions. However, direct adjustments to the boundaries of Secure world memory regions may affect the memory used by the Normal world. Therefore, memory management in TrustZone is always fixed during the boot stage. Besides, the minimum size of memory regions is 32KB [27], which may lead to memory waste.

CCA records the associated world of each granule in GPI. Therefore, it can achieve dynamic memory transitions between different worlds by modifying them during runtime. Regarding granularity, GPT supports configurations for the associated world of each memory region in 4KB, which is finer than TrustZone. To support validation for access from DMA devices, CCA extends GPC to SMMU. However, GPC in SMMU only supports the validation for the output PA, which means that access from DMA devices can only be blocked by the granularity of the world. For memory access management in finer granularity, CCA introduces Device Assignment (DA) [30]. With DA, a DMA device can be bound to a specific Realm VM and allowed only to access the memory region allocated to the Realm VM.

Compared to TrustZone, CCA optimizes memory utilization through dynamic memory allocation and finer granularity. Besides, since GPC is a part of MMU, each core can have its memory partition configuration. Conversely, the memory management provided by TZASC is shared by all cores. Therefore, CCA can support core-specific configuration, which TrustZone does not support. However, there is a limitation in the permission management in GPC. Unlike TZASC, the GPT can only determine which world each page belongs to and does not support separate configurations of read and write permissions.

B. Peripheral Management

Peripheral management represents the system's ability to adjust permission settings of peripherals to meet specific requirements. We evaluate this criterion of TrustZone and CCA from four aspects:

- **Dynamic Configuration:** The ability to change the security state of peripherals dynamically.
- **Minimal Granularity:** The smallest configurable size of a peripheral.
- **Peripheral Number:** The number of peripherals that can be configured.
- **Core-specific Configuration:** The ability to apply different configurations for different core.

In TrustZone, TZPC is used to configure the security state dynamically for each peripheral. However, the granularity of TZPC is limited to the whole peripheral. For example, all of a peripheral's registers are accessible to the Secure world if the peripheral's security state is set to Secure. Besides, the number of peripherals that TZPC can manage is limited. TZPC uses register bits to denote the associated world of each peripheral [28]. There are only three registers, and each register only has 8 bits for configuration. Therefore, TZPC can only manage 24 peripherals at most.

In Arm architecture, access to peripherals is achieved through Memory Mapped I/O (MMIO), allowing processors to access peripherals like it accesses main memory. Therefore, GPC can be used to check the access permissions from processors to peripherals. By modifying the GPI corresponding to the peripheral's physical address, the access permissions of each peripheral can be dynamically configured. Besides, there is no limitation on the number of peripherals that GPC can manage.

TABLE II: Summary for Comparison in Flexibility. "●" denotes corresponding features are supported, and "–" denotes corresponding features are not supported.

Criteria		**TrustZone**	**CCA**
Memory Management (§III-A)	Dynamic Allocation	●	●
	Minimal Granularity	32KB	4KB
	Memory Region Number	Limited	Unlimited
	R/W-separate Configuration	●	–
	Core-specific Configuration	–	●
Peripheral Management (§III-B)	Dynamic Configuration	●	●
	Peripheral Number	Limited	Unlimited
	Core-specific Configuration	–	●

C. Summary

The comparison results in terms of flexibility are summarized in Table II. Compared to TrustZone, CCA provides more flexibility in memory and peripheral management. CCA supports dynamic and finer-grained memory allocation. Besides, it can manage an unlimited number of memory regions and peripherals and support core-specific configurations. However, there is a limitation in the permission management in CCA, since GPC does not support separate configurations of read and write permissions.

IV. Security Comparison

A. Memory Isolation

Memory isolation means preventing the processor and DMA devices from illegally accessing memory. In TrustZone, TZASC ensures the isolation between the Normal world and the Secure world. Since it lies between the interconnect and the memory, it can block access from processors and DMA devices. In CCA, GPC is applied to check the output PA of the address translation in MMU and SMMU. Although the memory isolation mechanisms in TrustZone and CCA can be used to block illegal memory access, they have different characteristics.

First, TZASC can be configured by the software running in S-EL1/2. In this way, attackers can bypass the memory isolation mechanism once they hijack the TOS in TrustZone. In contrast, GPC can only be configured by the software running in EL3. Besides, TZASC does not support isolation between S-EL1/2 and EL3. Since the code and data in EL3 belong to the Secure world, attackers can exploit the software running in S-EL1/2 to affect the execution of *Monitor*. However, CCA moves the code and data belonging to EL3 from the Secure world to the Root world, which can mitigate this kind of security risks.

B. Memory Encryption

Memory encryption can protect code and data in memory from physical attacks. TrustZone hardware extensions do not include built-in memory encryption. Therefore, attackers can access to DRAM data in the Secure world through physical attacks such as the cold boot attack [34]. Although it can be implemented in software, there is usually a large performance overhead, and the original software needs to be modified [35], [36], [37]. Conversely, CCA offers hardware-assisted memory encryption through MPE, ensuring data integrity. In the design of CCA, except for the Normal world, the other three worlds must be enabled with encryption. A separate encryption key or tweak is used for each world, and spatial isolation is guaranteed with address tweaks [38]. Since the encryption can be applied to the Secure world, TrustZone can benefit from the memory encryption with MPE.

With MPE, although attackers can extract data from memory, they cannot understand the meaning of encrypted data since they do not know the keys. Nevertheless, MPE does not guarantee content erasure when the world change is applied to a granule. Explicit content scrubbing by software is necessary before transitioning.

C. Peripheral Isolation

Peripheral isolation means preventing the processor and DMA devices from illegally accessing peripherals. Blocking malicious access to some peripherals is necessary since private data may be stored there. For example, sensing systems use the existing or external sensors attached to mobile devices to capture various data types [39]. Passwords, fingerprints, and other sensitive data are also input from peripherals.

TrustZone uses TZPC and AXI interconnect to regulate the access to peripherals. TZPC configures the access permissions of peripherals, and AXI interconnect can then check the permissions with the access source. If the security state of processors or DMA devices does not match the permissions, AXI interconnect will block such invalid access. CCA leverages GPC in MMU to block invalid access to peripherals from processors. In Arm architecture, MMIO is used to support access to peripherals. By configuring the entries of the address of registers and buffers belonging to peripherals in GPT, CCA can block invalid access to peripherals. Similarly, invalid access from DMA devices can be blocked by GPC in SMMU.

Although both TrustZone and CCA support peripheral isolation, they have different security levels. Similar to TZASC, TZPC can be configured by the software in S-EL1, while GPT can only be modified by the software in EL3. In this way, attackers can bypass the peripheral isolation mechanism once they hijack the TOS.

D. Interrupts Isolation

An interrupt is a signal from hardware or software sent to the processor to indicate that an event has occurred. Malicious interrupts can interfere with the expected workflow of the processor. Manipulating an interrupt mechanism during the exploitation makes it easier to trigger these races, which is nearly impossible due to their unique requirement on execution orders [40]. Furthermore, the lack of interrupt isolation may hinder the deployment of TEEs to the real-time scenario [41],

[42]. Therefore, it is necessary for TEEs to ensure that interrupts can be routed correctly and securely.

TrustZone supports the isolation of secure interrupts and non-secure interrupts. In GICv3 affinity mode [29], the interrupt type is IRQ if the state of the processor matches the target of the interrupt, while the interrupt type is FIQ in other cases. IRQ can be handled by software in the current security state, while FIQ needs to be routed to software in EL3 for further processing. In this way, software in the Normal world cannot preempt secure interrupts.

For CCA, the hypervisor in the Normal world virtualizes interrupts for Realm VMs and then signals to the Realm through commands passed to RMM [21]. Attackers can pass malicious interrupts to Realm VMs by compromising the hypervisor. Then, unforeseen problems may happen when Realm VM handles malicious interrupts since it violates its design assumptions.

E. Hardware-assisted Attestation

The hardware-assisted attestation brings significant benefits to system security and integrity. It measures the system's state and provides assurance that the software running on the system has not been tampered with or modified since its initial trusted state. This capability is crucial in detecting unauthorized changes, such as malware injections and unauthorized updates. Moreover, hardware-assisted attestation can verify whether applications run on a platform that genuinely supports the required security features. It ensures that the VMs or applications are loaded and launched on the platform with the required security features supported.

TrustZone lacks hardware-assisted attestation [20], [43], whereas CCA offers this functionality for Realm VMs. Following is the process of the attestation for Realm VMs: At first, Realm VMs initiate an attestation request to Realm attestation service. Realm attestation service creates a Realm attestation token with information about the Realm boot state. Then, the Realm attestation service sends a request to the CCA platform attestation service. CCA platform attestation creates a Hardware Enforced Security (HES) host attestation token with information about the HES host boot state and a CCA platform boot token with information about the CCA platform boot state. Finally, these tokens are combined into a report. Users can request this report at any time. The hardware in CCA implements this feature by being bound to a unique identity and supporting the measurement of essential firmware.

F. Isolation in TLB and Cache

The Translation Lookaside Buffer (TLB) and cache are critical components in the memory hierarchy that enhance system performance. When the processor tries to access the memory, it first checks whether the translation result and data are in TLB and cache. However, since the hardware extensions for memory isolation are behind the TLB and cache, they cannot intercept access to the TLB and cache. Therefore, there is a need for TEEs to provide additional hardware mechanisms to ensure the security of TLB and cache.

With TrustZone enabled, each entry in TLB has an additional field to indicate whether the entry belongs to the Secure world or the Normal world. The entries for the same virtual address in different worlds are independent. When applications run in the Normal world, the MMU only searches the TLB entries belonging to the Normal world. Therefore, it can prevent applications in the Normal world from accessing the Secure world. Similarly, TrustZone divides the cache into two parts by extending an NS bit in the cache line [44]. In this way, the processor can distinguish cache lines from different worlds and fetch data correctly.

CCA extends the TLB and cache with an additional bit to support identifying the Realm world and the Root world. Besides, the TLB in CCA caches the translation of virtual addresses to physical addresses and the GPT entries. Since the CCA supports dynamic memory allocation, the GPT entries may be updated at any time. If the GPT entries are not updated in time, attackers can bypass the memory isolation mechanism. Therefore, there is a need to ensure the consistency of entries in TLB. CCA does not provide hardware mechanisms to invalidate TLB entries automatically. Instead, CCA provides additional instructions for the firmware developers to invalidate cached copies of GPT entries from TLBs based on physical addresses [45]. When a region of memory in the Normal world can be transferred to the Realm world, the firmware must invalidate the TLB entries, ensuring that software running in the Normal world cannot access the memory region that has been moved.

TABLE III: Summary for Comparison in Security. "●" denotes corresponding features are supported, and "–" denotes corresponding features are not supported.

Criteria		**TrustZone**	**CCA**
Memory Isolation (§IV-A)	Access Control for processors	●	●
	Access Control for DMA	●	●
	Isolation between S-EL1/2 and EL3	–	●
	Level to Configure	S-EL1/2	EL3
Memory Encryption (§IV-B)	Hardware-assisted Encryption	–	●
Peripheral Isolation (§IV-C)	Access Control for processors	●	●
	Access Control for DMA	●	●
	Level to Configure	S-EL1/2	EL3
Interrupt Isolation (§IV-D)	Individual Interrupt for TEE	●	–
Attestation (§IV-E)	Hardware-assisted Attestation	–	●
TLB and Cache (§IV-F)	Isolation in TLB and Cache	●	●

G. Summary

The comparison results in terms of security are summarized in Table III. CCA provides stronger security guarantees than TrustZone in most aspects. The isolation mechanism in CCA can only be configured by the software running in EL3 and supports to isolate EL3 from other worlds. Besides, it supports hardware-assisted memory encryption and hardware-assisted attestation. However, since interrupts for the Real world in

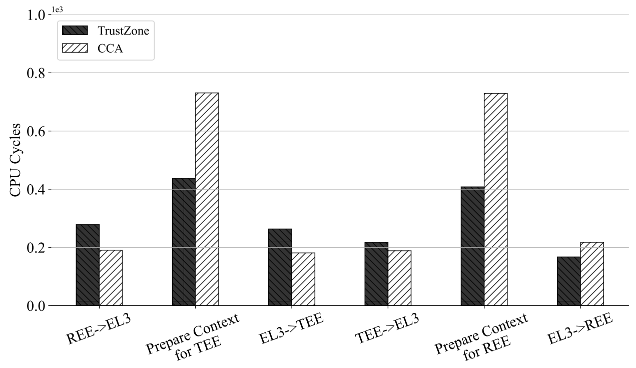

Fig. 4: Performance overhead in different stages during context switch.

CCA are virtualized by the hypervisor, it may introduce unforeseen problems when handling malicious interrupts.

V. Performance Evaluation

A. Environment Setup

We use a Juno-R2 development board to evaluate the performance of benchmarks on TrustZone, which has 2 Cortex-A72 (1.2GHz) big cores and 4 Cortex-A53 (950MHz) small cores. All these cores are Armv8-A and support TrustZone extensions. The firmware we use to evaluate is TF-A v2.3 and OPTEE v3.6.0, and the Linux kernel version of the host OS is 5.3.0.

To ensure a fair comparison of performance between CCA and TrustZone, we maintain consistency by using the Juno-R2 development board for evaluating benchmark performance on CCA. The firmware we use is based on TF-A v2.3 and TF-RMM v0.2.0. The Linux kernel version of the host OS is 5.3.0, and we modify its KVM to support CCA. To host Realm VMs, we employ a modified version of Kvmtool [46], which Arm officially provides. We use the Performance Monitoring Unit (PMU) to count CPU cycles. Besides, since the test results may be unstable when enabling big and small cores, we only enable 4 Cortex-A53 small cores.

B. Microbenchmarks

World Switch. In practice, some tasks require collaboration between the TEE and the REE. This collaboration demands a context switch between the TEE and the REE, making it vital to evaluate the efficiency of this switch. If the context switch wastes too much time, the TEE may not be well-suited for certain scenarios. To address this concern, we break down the context switch process into multiple steps and measure the cost of each step. The workflow involved in the context switch process is: (1) The application in the Normal world begins the context switch by invoking SMC instruction. It jumps to EL3 and starts at the address of the interrupt vector table. The stack pointer (SP) is replaced with `SP_EL3`. After that, it saves general-purpose registers from register x0 to register x29. Because `SP_EL3` only stores the `cpu_context` structure, it needs to restore real runtime stack value from `SP_EL3`. This stack value is put into `SP_EL0`. Then, it goes to the corresponding SMC handler, which is known as the dispatcher. (2) At the corresponding dispatcher in EL3, it saves the system registers of the source security state. Then, it restores the system registers of the target security state. At last, it sets `cpu_context` for the target security state. (3) Before leaving EL3, it saves `SP_EL0` and restores `SP_EL3`. After that, it can restore general-purpose registers from `SP_EL3`. When all the registers are prepared, it uses the `eret` instruction to leave EL3. (4) When the required task is finished in the TEE, it issues an SMC call to return to EL3. (5) Similar to step 2, the dispatcher saves and restores system registers and sets the next `cpu_context`. (6) Finally, it returns control to the REE.

In the context switch microbenchmark, an SMC call is used to trigger the context switch. Figure 4 shows the time cost of each step in the context switch process. The cost of roundtrip transitions between EL3 and the REE and between EL3 and the TEE are similar. This is because issuing an SMC call to the EL3 firmware and using the eret instruction to return to the TEE and REE incur no additional overhead in TrustZone and CCA. The key distinction in the context switch between TrustZone and CCA lies in the cost of saving and restoring system registers. In CCA, system registers related to EL1 and EL2 are saved and restored in EL3. As for TrustZone, since TOSs such as OPTEE run at S-EL1, only the system registers associated with EL1 are saved and restored in EL3.

C. Benchmarks

CPU Intensive Workload Comparison. In this test, we choose nbench [47] to evaluate the impact of CCA and TrustZone on performance. Nbench is a testing program used to evaluate the performance of computer systems. It includes a group of routines that simulate common computer tasks, such as numeric sort and LU decomposition.

As shown in Figure 5, the performance overhead of CCA is similar to TrustZone in most CPU intensive cases. One reason is that the major work of these benchmarks is finished in user space. Therefore, the context switch caused by running test routines in Realm VMs is negligible and does not introduce much overhead. Besides, the cache also reduces the performance overhead. Nbench mainly tests the computing capability of the CPU. Since the size of the data and instructions it uses is small enough to be stored in the cache, the two-stage translation that comes from virtualization will not significantly impact it.

Memory Intensive Workload Comparison. Besides CPU intensive workloads, we also use STREAM [48] benchmark and Random Access benchmark to evaluate the performance of CCA and TrustZone in memory intensive workloads. The STREAM benchmark is a synthetic benchmark program designed to measure the sustainable memory bandwidth and the corresponding computation rate. As for the Random Access benchmark, it measures the random access performance of memory.

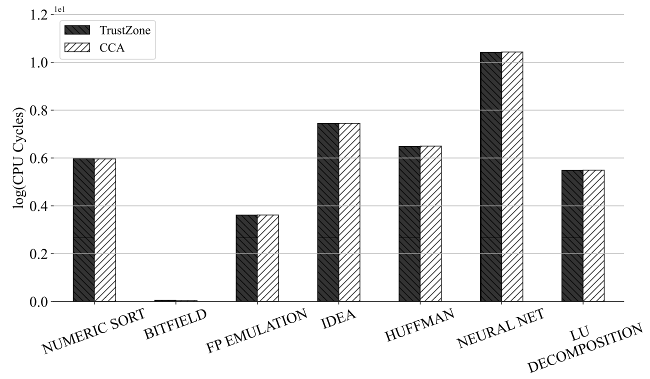

Fig. 5: Performance overhead on CPU intensive workloads.

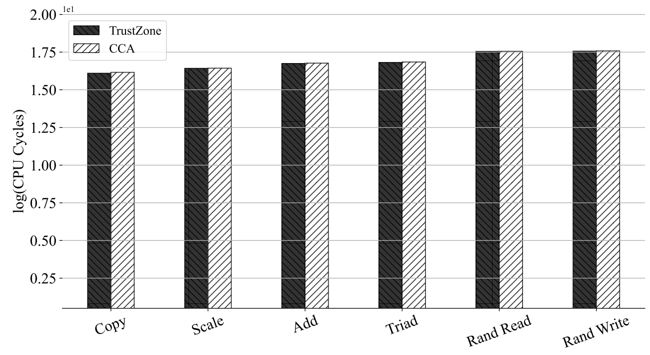

Fig. 6: Performance overhead on memory intensive workloads.

Figure 6 shows the performance overhead of TrustZone and CCA in each case. TrustZone is slightly faster than CCA on average. In both STREAM and Random Access, we only evaluate the performance overhead in user space. The virtualization mainly causes the performance gap between TrustZone and CCA. In CCA, it requires a two-stage translation to access the memory. However, in TrustZone, the memory access is translated in one stage. In addition, the benchmark utilizes a memory size that exceeds cache size several times, effectively reducing cache hits and minimizing the impact of cache. As a result, TrustZone outperforms CCA in memory-intensive workloads.

I/O Intensive Workload Comparison. We also design an application to evaluate the performance overhead in I/O intensive workloads. The workflow of this application is as follows: First, the application reads a file from the disk and loads the content to the TEE. Then, the application decrypts the file content and calculates the hash value based on the content. Finally, the application encrypts the result and writes it back to the filesystem.

Figure 7 shows that the performance overhead of CCA is much lower than TrustZone because OP-TEE cannot read and write files directly. When the trusted application (TA) in the Secure world requires file access, it will invoke the CA in the Normal world to complete the file access. Then, the TA copies the file content from the shared buffer to the secure

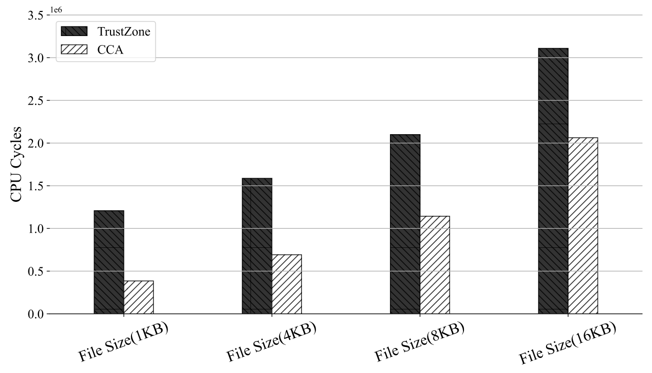

Fig. 7: Performance overhead on I/O intensive workloads. The x-axis represents the size of the file read by the application.

memory and finishes the tasks. In CCA, applications running in Realm VMs can access files through VirtIO, which is more efficient than the method used in TrustZone. Therefore, CCA outperforms TrustZone in I/O intensive workloads.

VI. Limitation Analysis

A. *TrustZone's Limitations*

TrustZone works well at protecting sensitive data and code. However, due to its design choices, it still has some limitations. These limitations can be divided into two categories based on their root causes. The first category (**L1-4**) is caused by the deployment and security considerations of vendors. For example, to ensure security, vendors usually do not open the Secure world to third-party developers and limit the interfaces exposed to applications. The second category (**L5-7**) is caused by the design of TrustZone hardware extensions. Specifically, they are caused by the lack of consideration for security threats (**L5, 6**) and use cases (**L7**).

Limitation 1: TAs are not protected from the system software in the Secure software. The design goal of TrustZone is to isolate the Secure world from the Normal world. Therefore, TrustZone can protect TAs from the system software in the Normal world but does not protect them from the system software in the Secure world [22]. The system software in the Secure world, like the TOS, has a large code base since they need to implement enough secure services to host applications independently, which leads to large potential vulnerabilities that attackers may exploit [20], [49]. In practice, not all the components of system software needs access to memory regions used by applications [50], [51]. Ensuring different access permissions among components becomes crucial to mitigate threats from compromised system software components within the Secure world.

Limitation 2: The Secure world of TrustZone is not open to third-party developers. Vulnerabilities in TAs may expose new attack surfaces for attackers to compromise the whole system [52], [53]. For example, attackers can use a vulnerable TA as a trampoline to gain control of the Linux Kernel [54]. Therefore, vendors control and restrict access to the Secure world to ensure security [55], [20]. However, such a design choice limits the use cases of TrustZone. The restrictive deployment policy makes it difficult for third-party developers to take advantage of hardware features provided by TrustZone.

Limitation 3: The Secure world of TrustZone lacks compatibility. To keep TCB as small as possible, TOS is designed to be lightweight and exposes only a small number of interfaces to applications, making it different from the OS in the Normal world. Therefore, the development of TAs follows a different process from the development of applications in the Normal world [56], [57], [58]. These compatibility issues can result in costly and time-consuming efforts to port legacy applications from the Normal world into the Secure world. Besides, different vendors may have different implementations of the TOS [59], which leads to additional effort to port trusted applications.

Limitation 4: Extending TrustZone for GPUs and other accelerators introduces a large TCB. Developers rely on GPUs and accelerators to achieve better performance for workloads like AI and graphics-intensive applications. These accelerators often process sensitive data, such as personal information and trade secrets. A straightforward approach is to port workloads and corresponding device drivers to the Secure world. However, it results in excessive secure memory usage and introduces a large TCB [60]. Therefore, using GPU and accelerators for secure computation while keeping small TCB low is essential to cover a broader range of TrustZone.

Limitation 5: The system software in the Secure world has unrestricted access to the whole system memory. In TrustZone, TZASC is configurable by software running in S-EL1/2 [27]. Therefore, attackers can exploit vulnerabilities in the system software in the Secure world and configure the TZASC to turn off memory isolation [61]. Then, they can access any memory region and compromise the whole system.

Limitation 6: TrustZone lacks support for memory encryption. TrustZone hardware extensions do not include a component to encrypt memory. This limitation makes it difficult to protect memory from physical attacks, such as cold boot attacks [34], [62], bus monitoring attacks [63], and DMA attacks [64]. Through physical attacks, attackers can access the memory and retrieve sensitive data without exploiting software vulnerabilities.

Limitation 7: TrustZone lacks flexibility in memory management. TrustZone partition memory into different regions and configure their properties through TZASC. However, the number of configurable regions is limited [27]. TZASC cannot deal with enforced isolation between memory regions when physical memory is heavily fragmented [65]. Besides, the minimum size of memory regions is 32KB [27], affecting the efficiency of memory utilization.

Projects for Addressing TrustZone's Limitations. In response to these limitations, researchers have proposed many projects. We list the approaches used by these projects and explain them in Table IV. Besides, we list the limitations addressed by these projects and the corresponding drawbacks of the projects in Table V.

For Limitation 1, both vTZ [66] and Twinvisor [67] address it by partitioning the system software in the Secure world into two components and running them in different execution environments. Limitation 2 is addressed by constructing an additional execution environment for third-party applications. Projects for addressing this limitation adopt a secure monitor to ensure isolation for different execution environments. However, the mechanisms behind these secure monitors are different. For TrustICE [70], the secure monitor

TABLE IV: Approaches to address TrustZone's limitations and their description.

	Approach	Description
Software	**A1**: Trusted Runtime	Ensure isolation for applications; provide critical system servicesand forward unhandled exceptions to the host OS
	A2: Decoupled Hypervisor	Ensure isolation for OSs; provide critical system services and forward unhandled exceptions to the host hypervisor
	A3: Secure Partition Monitor	Ensure isolation for OSs and work independently
	A4: Secure Monitor	Ensure isolation for different execution environments (including EL0, EL1, and EL2)
	A5: Record and Replay	Record the required instructions for secure tasks outside the TEE and execute them inside the TEE
	A6: Binary Scanning	Remove specific instructions that may break isolation
	A7: Memory Allocation Strategy	Allocate memory using specific schemes, such as continuous memory allocation (CMA)
Hardware	**A8**: Two Stage Translation	Translate VA to Intermediate PA (IPA) in Stage-1 and then translate IPA to PA in Stage-2
	A9: On-chip Memory	Integrated with the chip and non-removable
	A10: Resource Domain Controller	Restrict physical access to a specific resource and can differentiate between multiple bus masters
	A11: Modified TZASC	Extend TZASC to support core-specific configurations

TABLE V: Projects for Addressing TrustZone's Limitations. **Target**: "●" denotes this limitation is addressed, "◐" denotes this limitation is addressed partially, and "–" denotes this limitation is not addressed. "**An**" denotes this limitation is addressed with Approach n in Table IV. **Drawback**: "●" denotes the project has this drawback, "◐" denotes the project has this drawback in certain cases, and "–" denotes the project is without this drawback.

		vTZ [66]	Twinvisor [67]	TLR [68]	TrustShadow [69]	SecDeep [60]	TrustICE [70]	TEEv [59]	PrOS [65]	GR-T [71]	Strongbox [72], [73]	PrivateZone [74]	MyTEE [75]	CryptMe [35]	Minimal Kernel [37]	SecTEE [36]	ReZone [61]	SANCTUARY [55]	Hafnium [76]
Target	Limitation 1	● **A2**	● **A2**	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	Limitation 2	–	–	–	–	–	● **A4**	–	–	–	–	● **A4**	● **A4**	–	–	–	–	● **A11**	–
	Limitation 3	–	–	● **A1**	● **A1**	–	–	● **A3**	–	–	–	–	–	● **A1**	–	–	–	–	–
	Limitation 4	–	–	–	–	● **A1**	–	–	–	● **A5**	● **A8**	–	–	–	–	–	–	–	–
	Limitation 5	–	–	–	–	–	–	● **A3,6**	● **A3,6**	–	–	–	–	–	–	–	● **A10**	–	● **A3**
	Limitation 6	–	–	–	–	–	–	–	–	–	–	–	–	● **A9**	● **A9**	● **A9**	–	–	–
	Limitation 7	–	◐ **A7**	–	–	–	–	–	◐ **A7**	–	–	–	–	–	–	–	–	–	–
Drawback	Software Modification	–	–	●	–	–	–	●	●	–	◐	◐	◐	–	–	–	–	–	–
	Hardware Modification	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	●	–
	Large Performance Overhead	●	–	●	–	–	●	–	–	◐	–	–	–	●	●	●	–	–	–
	Large Memory Overhead	–	–	–	–	–	–	–	–	●	–	–	–	–	–	–	–	–	–
	Lack of Generality	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	●	–	–
	Large TCB	–	–	–	–	●	–	–	–	–	–	–	–	–	–	–	–	–	●

ensures isolation by time slicing. It prevents invalid access to the protected memory by hanging out software running in other execution environments. However, such a mechanism introduces a large performance overhead and is unsuitable for multicore processors. Other secure monitors ensure spatial memory isolation by configuring additional hardware features. PrivateZone [74] and MyTEE [75] use the two stage translation, and SANCTUARY [55] uses the modified TZASC. Limitation 3 can be addressed by providing a compatibility layer for applications. TLR [68] supports applications written in .NET by porting the language runtime to the Secure world, and TrustShadow [69] supports unmodified Linux applications by forwarding system calls to the Normal world. Limitation 4 can be addressed by placing the memory used by GPUs in the Secure world [60]. However, porting the GPU driver causes a large increase in TCB. GT-R [71] avoids software porting by recording executed instructions of GPU drivers in the cloud and replaying them in the Secure world. However, this approach introduces a large performance overhead and memory overhead. Strongbox [72], [73] restricts access to memory used by GPUs through two stage translation. Besides, it uses a trusted runtime to forward requests related to GPU drivers, which avoids introducing a large TCB.

Limitation 5 is caused by the architecture design of TrustZone. TEEv [59], PrOS [65] and Hafnium [76] address limitation by introducing a secure hypervisor. However, because Secure EL2 is not supported in TEEv and PrOS, they must conduct binary scanning of TOSs to ensure the absence of instructions that could compromise the isolation guaranteed by the provided memory mapping. ReZone [61] achieves the same goal in another way. It uses Resource Domain Controller (RDC) to restrict access from processors to memory. For Limitation 6, CryptMe [35], Minimal Kernel [37] and SecTEE [36] address it by on-chip memory, which is integrated with the chip and non-removable. They encrypt contents in the external memory and decrypt them when they are loaded into the on-chip memory. However, these projects introduce a large performance overhead. Limitation 7 is due to the design of TZASC. PrOS [65] and Twinvisor [67] improve efficiency of memory utilization by well-designed memory management schemes. However, they can only address this limitation partially. As long as TZASC is still relied on to achieve memory isolation, problems like coarse granularity in memory management cannot be solved completely.

Despite these projects having made progress in solving the limitations of TrustZone, they usually focus on specific aspects of these limitations and have their drawbacks. Although combining approaches used by these projects can address most of the limitations, integrating them into one project is challenging. In addition, such complex designs may introduce new challenges in security and performance. Therefore, addressing these limitations without any side effects can be challenging without support from new hardware extensions designed for TEEs.

B. CCA's Impacts and Limitations

Many of the limitations of TrustZone resulting from hardware extension design can be resolved through new hardware features in CCA (**L5,6,7**). Besides, CCA introduces the Realm world, allowing developers to run their applications in VM-level TEEs (**L2,3**). Furthermore, CCA has different design requirements for the system software in the Realm world, which can also address the limitations caused by system software design (**L1**).

However, there are still limitations that CCA does not address. For example, CCA also does not consider the TCB size increased by drivers of GPU and other accelerators (**L4**). Besides, there are some unique limitations in CCA. First, the approach proposed in CCA to address **L1** requires modifications to existing system software, which increases the burden on vendors. Second, CCA provides confidential execution environments for developers at the VM level. The large codebase introduced by the guest OS exposes a large attack surface. However, some ways like unikernel [77] can mitigate this issue. Moreover, GPC does not provide fine-grained control in R/W permission for memory regions.

VII. Related Works

Several surveys mentioned different aspects of TrustZone. Sabt *et al.* [78] propose a refined definition of TEEs and compare TrustZone-assisted TEEs using the proposed definition. Ngabonziza *et al.* [79] discuss Arm architectures supporting TrustZone, including Armv6, Armv7, and Armv8. It introduces the details, such as processor states and system registers of these architectures, and reviews how they implement TrustZone in the hardware and software. Ning *et al.* [80] and Cerdeira *et al.* [20] discuss potential security threats to TrustZone and how to mitigate them.

Some studies compare TrustZone with other TEEs. Maene *et al.* [81] discuss the details of twelve hardware-assisted TEEs from industry and academia and compare their security properties and architecture features. Although these studies have discussed Arm TrustZone, it has been several years since these papers were published. There is a gap between the content discussed in these papers and the state-of-art Arm architecture. We hope our work can fill this gap. Pinto *et al.* [1] compare different TrustZone-assisted TEE systems regarding their type, TCB size, and whether they support secure user interface or storage. They also compare several TrustZone-assisted virtualization solutions and classify them according to the number of VMs they can support. Besides TrustZone, other TEE hardware technologies, including Intel Software Guard Extensions (Intel SGX) [82] and AMD Secure Encrypted Virtualization (SEV) [83], are also discussed in this paper. Intel SGX allows developers to create an enclave in the user space, which can be used to protect the code and data from system software. AMD SEV ensures the integrity and confidentiality of the VM's memory by encrypting the memory of the VM with a unique key. In this way, the hypervisor can only manage the execution of the VM but cannot access its memory. Demigha *et al.* [84] present, analyze, and compare four major industrial-scale commercial hardware-assisted TEEs in the cloud market. These four TEEs are Intel TXT, Arm TrustZone, AMD SEV, and Intel SGX. In this paper, the authors measure these TEEs from security, functionality, and deployment. They also analyze the differences between TEEs in the specific scenario of cloud computing.

VIII. Conclusion

In this paper, we present a comparison study on TrustZone and CCA. We first introduce the architecture features of TrustZone and CCA. Then, we compare them regarding flexibility, security, and performance. Following this, we analyze limitations in TrustZone and discuss CCA's impacts on TrustZone's limitations and its unique limitations. We hope this paper can help the community better understand their different characteristics and application scenarios.

Acknowledgments

We would like to thank the anonymous reviewers for their insightful comments. This work is partly supported by the National Natural Science Foundation of China under Grant No.62372218, Shenzhen Science and Technology Program under Grant No.SGDX20201103095408029, and Ant Group.

References

[1] S. Pinto and N. Santos, "Demystifying Arm TrustZone: A Comprehensive Survey," *ACM computing surveys (CSUR)*, vol. 51, no. 6, pp. 1–36, 2019.
[2] P. Sparks, "The route to a trillion devices," *White Paper, ARM*, 2017.
[3] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and Privacy Challenges in Industrial Internet of Things," in *Proceedings of the 52nd annual design automation conference*, 2015, pp. 1–6.
[4] C. Spensky, J. Stewart, A. Yerukhimovich, R. Shay, A. Trachtenberg, R. Housley, and R. K. Cunningham, "SoK: Privacy on Mobile Devices-It's Complicated," *Proc. Priv. Enhancing Technol.*, vol. 2016, no. 3, pp. 96–116, 2016.
[5] Z. Ling, K. Liu, Y. Xu, Y. Jin, and X. Fu, "An End-to-End View of IoT Security and Privacy," in *GLOBECOM 2017-2017 IEEE Global Communications Conference.* IEEE, 2017, pp. 1–7.
[6] Z. Zhang, H. Zhang, Z. Qian, and B. Lau, "An investigation of the android kernel patch ecosystem," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3649–3666.
[7] Z. Lin, Y. Wu, and X. Xing, "DirtyCred: Escalating Privilege in Linux Kernel," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1963–1976.
[8] F. Zhang and H. Zhang, "Sok: A study of using hardware-assisted isolated execution environments for security," in *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, 2016, pp. 1–8.
[9] S. Mofrad, F. Zhang, S. Lu, and W. Shi, "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*, 2018, pp. 1–8.
[10] T. Alves, "Trustzone: Integrated hardware and software security," *Information Quarterly*, vol. 3, pp. 18–24, 2004.
[11] ARM, "ARM Security Technology Building a Secure System using TrustZone Technology," https://developer.arm.com/documentation/PRD29-GENC-009492/latest/, 2009.
[12] Android, "Android keystore system," https://developer.android.com/training/articles/keystore, 2022.
[13] Huawei, "Privacy," "https://consumer.huawei.com/au/sustainability/privacy/", 2023.
[14] Qualcomm, "Guard your data with the qualcomm snapdragon mobile platform," "https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/guard_your_data_with_the_qualcomm_snapdragon_mobile_platform2.pdf", 2019.
[15] Z. Ning, C. Wang, Y. Chen, F. Zhang, and J. Cao, "Revisiting arm debugging features: Nailgun and its defense," *IEEE Transactions on Dependable and Secure Computing*, 2021.
[16] D. Xie, Y. Hu, and L. Qin, "An Evaluation of Serverless Computing on X86 and ARM platforms: Performance and Design Implications," in *2021 IEEE 14th International Conference on Cloud Computing (CLOUD).* IEEE, 2021, pp. 313–321.
[17] S. Xu, A. Shafi, H. Subramoni, and D. K. Panda, "Arm meets Cloud: A Case Study of MPI Library Performance on AWS Arm-based HPC Cloud with Elastic Fabric Adapter," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* IEEE, 2022, pp. 449–456.
[18] A. Pellegrini, N. Stephens, M. Bruce, Y. Ishii, J. Pusdesris, A. Raja, C. Abernathy, J. Koppanalil, T. Ringe, A. Tummala *et al.*, "The Arm Neoverse N1 Platform: Building Blocks for the Next-Gen Cloud-to-Edge Infrastructure SoC," *IEEE Micro*, vol. 40, no. 2, pp. 53–62, 2020.
[19] A. Pellegrini, "Arm Neoverse N2: Arm's 2nd generation high performance infrastructure CPUs and system IPs," in *2021 IEEE Hot Chips 33 Symposium (HCS).* IEEE, 2021, pp. 1–27.
[20] D. Cerdeira, N. Santos, P. Fonseca, and S. Pinto, "SoK: Understanding the Prevailing Security Vulnerabilities in TrustZone-assisted TEE Systems," in *2020 IEEE Symposium on Security and Privacy (SP).* IEEE, 2020, pp. 1416–1432.
[21] ARM, "Introducing Arm Confidential Compute Architecture," https://developer.arm.com/documentation/den0125/, 2022.
[22] ARM, "Arm CCA Security Model," https://developer.arm.com/documentation/DEN0096, 2021.
[23] "Arm fixed virtual platforms." https://developer.arm.com/tools-and-software/simulation-models/fixed-virtual-platforms, 2021.
[24] X. Li, X. Li, C. Dall, R. Gu, J. Nieh, Y. Sait, and G. Stockwell, "Design and Verification of the Arm Confidential Compute Architecture," in *Proceedings of the 16th USENIX Symposium on Operating Systems Design and Implementation*, 2022, pp. 465–484.
[25] Y. Zhang, Y. Hu, Z. Ning, F. Zhang, X. Luo, H. Huang, S. Yan, and Z. He, "SHELTER: Extending Arm CCA with Isolation in User Space," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.
[26] ARM, "Learn the architecture: Trustzone for aarch64," https://developer.arm.com/documentation/102418/0101/What-is-TrustZone-, 2021.
[27] ARM, "ARM CoreLink TZC-400 TrustZone Address Space Controller Technical Reference Manual," https://developer.arm.com/documentation/ddi0504/latest/, 2014.
[28] ARM, "PrimeCell Infrastructure AMBA 3 TrustZone Protection Controller (BP147)," https://developer.arm.com/documentation/dto0015/latest/, 2004.
[29] ARM, "GICv3 and GICv4 Software Overview," https://developer.arm.com/documentation/dai0492/latest, 2016.
[30] ARM, "Arm System Memory Management Unit Architecture Specification," https://developer.arm.com/documentation/ihi0070, 2023.
[31] "Trusted-Firmware-A," https://git.trustedfirmware.org/TF-A/trusted-firmware-a.git/, 2022.
[32] "TF-RMM," https://github.com/TF-RMM/tf-rmm, 2022.
[33] "Linux-CCA," https://gitlab.arm.com/linux-arm/linux-cca, 2022.
[34] L. W. Remember, "Cold Boot Attacks on Encryption Keys," in *2008 USENIX Security Symposium*, vol. 21, 2008.
[35] C. Cao, L. Guan, N. Zhang, N. Gao, J. Lin, B. Luo, P. Liu, J. Xiang, and W. Lou, "CryptMe: Data leakage prevention for unmodified programs on ARM devices," in *Research in Attacks, Intrusions, and Defenses: 21st International Symposium, RAID 2018, Heraklion, Crete, Greece, September 10-12, 2018, Proceedings 21.* Springer, 2018, pp. 380–400.
[36] S. Zhao, Q. Zhang, Y. Qin, W. Feng, and D. Feng, "SecTEE: A Software-based Approach to Secure Enclave Architecture Using TEE," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1723–1740.
[37] S. Zhao, Q. Zhang, Y. Qin, W. Feng, and D. Feng, "Minimal Kernel: An Operating System Architecture for TEE to Resist Board Level Physical Attacks," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 105–120.
[38] ARM, "Arm Realm Management Extension (RME) System Architecture," https://developer.arm.com/documentation/den0129/ad, 2022.
[39] H. Janjua, W. Joosen, S. Michiels, and D. Hughes, "Trusted operations on sensor data," *Sensors*, vol. 18, no. 5, p. 1364, 2018.
[40] Y. Lee, C. Min, and B. Lee, "ExpRace: Exploiting Kernel Races through Raising Interrupts," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2363–2380.

[41] S. Pinto, J. Pereira, T. Gomes, A. Tavares, and J. Cabral, "LTZVisor: TrustZone is the Key," in *29th Euromicro Conference on Real-Time Systems (ECRTS 2017).* Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[42] J. Wang, A. Li, H. Li, C. Lu, and N. Zhang, "RT-TEE: Real-time System Availability for Cyber-physical Systems using ARM TrustZone," in *2022 IEEE Symposium on Security and Privacy (SP).* IEEE Computer Society, 2022, pp. 1573–1573.

[43] S. Matetic, M. Schneider, A. Miller, A. Juels, and S. Capkun, "DelegaTEE: Brokered Delegation Using Trusted Execution Environments," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1387–1403.

[44] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices," *Cryptology ePrint Archive*, 2016.

[45] ARM, "The Realm Management Extension (RME), for Armv9-A," https://developer.arm.com/documentation/ddi0615/latest, 2022.

[46] "kvmtool-cca," https://gitlab.arm.com/linux-arm/kvmtool-cca, 2022.

[47] F. Mayer, "Linux/Unix nbench," "https://www.math.utah.edu/~mayer/linux/bmark.html", 2017.

[48] J. D. McCalpin, "Stream: Sustainable memory bandwidth in high performance computers," University of Virginia, Charlottesville, Virginia, Tech. Rep., 1991-2007, a continually updated technical report. http://www.cs.virginia.edu/stream/. [Online]. Available: http://www.cs.virginia.edu/stream/

[49] F. Khalid and A. Masood, "Vulnerability analysis of Qualcomm Secure Execution Environment (QSEE)," *Computers & Security*, vol. 116, p. 102628, 2022.

[50] S.-W. Li, J. S. Koh, and J. Nieh, "Protecting Cloud Virtual Machines from Hypervisor and Host Operating System Exploits," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1357–1374.

[51] A. Van't Hof and J. Nieh, "BlackBox: A Container Security Monitor for Protecting Containers on Untrusted Operating Systems," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2022.

[52] D. Shen, "Exploiting trustzone on android," *Black Hat USA*, vol. 2, pp. 267–280, 2015.

[53] S. Wan, M. Sun, K. Sun, N. Zhang, and X. He, "RusTEE: Developing Memory-Safe ARM TrustZone Applications," in *Annual Computer Security Applications Conference*, 2020, pp. 442–453.

[54] laginimaineb, "War of the Worlds - Hijacking the Linux Kernel from QSEE," https://bits-please.blogspot.com/2016/05/war-of-worlds-hijacking-linux-kernel.html, 2016.

[55] F. Brasser, D. Gens, P. Jauernig, A.-R. Sadeghi, and E. Stapf, "SANCTUARY: ARMing Trustzone with User-space Enclaves," in *Proceedings of the 26th Annual Network and Distributed System Security Symposium*, 2019.

[56] GlobalPlatform, "Tee management framework including asn.1 profile," 2020. [Online]. Available: https://globalplatform.org/specs-library/tee-management-framework-including-asn1-profile-1-1-2/

[57] GlobalPlatform, "Tee client api specification," 2010. [Online]. Available: https://globalplatform.org/specs-library/tee-client-api-specification/

[58] GlobalPlatform, "Tee internal core api specification," 2021. [Online]. Available: https://globalplatform.org/specs-library/tee-internal-core-api-specification/

[59] W. Li, Y. Xia, L. Lu, H. Chen, and B. Zang, "TEEv: Virtualizing Trusted Execution Environments on Mobile Platforms," in *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, 2019, pp. 2–16.

[60] R. Liu, L. Garcia, Z. Liu, B. Ou, and M. Srivastava, "SecDeep: Secure and Performant On-device Deep Learning Inference Framework for Mobile and IoT Devices," in *Proceedings of the International Conference on Internet-of-Things Design and Implementation*, 2021, pp. 67–79.

[61] D. Cerdeira, J. Martins, N. Santos, and S. Pinto, "ReZone: Disarming TrustZone with TEE Privilege Reduction," in *Proceedings of the 31st USENIX Security Symposium*, 2022, pp. 2261–2279.

[62] T. Müller and M. Spreitzenbarth, "FROST: Forensic Recovery of Scrambled Telephones," in *Applied Cryptography and Network Security: 11th International Conference, ACNS 2013, Banff, AB, Canada, June 25-28, 2013. Proceedings 11.* Springer, 2013, pp. 373–388.

[63] M. G. Kuhn, "Cipher Instruction Search Attack on the Bus-Encryption Security Microcontroller DS5002FP," *IEEE Transactions on Computers*, vol. 47, no. 10, pp. 1153–1157, 1998.

[64] M. Gross, N. Jacob, A. Zankl, and G. Sigl, "Breaking TrustZone Memory Isolation through Malicious Hardware on a Modern FPGA-SoC," in *Proceedings of the 3rd ACM Workshop on Attacks and Solutions in Hardware Security Workshop*, 2019, pp. 3–12.

[65] D. Kwon, J. Seo, Y. Cho, B. Lee, and Y. Paek, "PrOS: Light-weight privatized se cure OSes in ARM TrustZone," *IEEE Transactions on Mobile Computing*, vol. 19, no. 6, pp. 1434–1447, 2019.

[66] Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, and H. Guan, "vTZ: Virtualizing ARM TrustZone," in *Proceedings of the 26th USENIX Security Symposium*, 2017, pp. 541–556.

[67] D. Li, Z. Mi, Y. Xia, B. Zang, H. Chen, and H. Guan, "Twinvisor: Hardware-isolated Confidential Virtual Machines for ARM," in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 638–654.

[68] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using ARM TrustZone to Build a Trusted Language Runtime for Mobile Applications," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, 2014.

[69] L. Guan, P. Liu, X. Xing, X. Ge, S. Zhang, M. Yu, and T. Jaeger, "TrustShadow: Secure Execution of Unmodified Applications with ARM TrustZone," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, 2017, pp. 488–501.

[70] H. Sun, K. Sun, Y. Wang, J. Jing, and H. Wang, "TrustICE: Hardware-Assisted Isolated Computing Environments on Mobile Devices," in *Proceedings of the 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks.* IEEE, 2015, pp. 367–378.

[71] H. Park and F. X. Lin, "Safe and Practical GPU Computation in TrustZone," in *Proceedings of the Eighteenth European Conference on Computer Systems*, 2023, pp. 505–520.

[72] Y. Deng, C. Wang, S. Yu, S. Liu, Z. Ning, K. Leach, J. Li, S. Yan, Z. He, J. Cao *et al.*, "Strongbox: A GPU TEE on Arm Endpoints," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 769–783.

[73] C. Wang, Y. Deng, Z. Ning, K. Leach, J. Li, S. Yan, Z. He, J. Cao, and F. Zhang, "Building a lightweight trusted execution environment for arm gpus," *IEEE Transactions on Dependable and Secure Computing*, no. 01, pp. 1–16, 2023.

[74] J. Jang, C. Choi, J. Lee, N. Kwak, S. Lee, Y. Choi, and B. B. Kang, "PrivateZone: Providing a Private Execution Environment Using ARM TrustZone," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 5, pp. 797–810, 2016.

[75] S.-K. Han and J. Jang, "MyTEE: Own the Trusted Execution Environment on Embedded Devices." in *NDSS*, 2023.

[76] "Hafnium architecture," https://hafnium.googlesource.com/hafnium/+/HEAD/docs/Architecture.md, 2021.

[77] A. Madhavapeddy, R. Mortier, C. Rotsos, D. Scott, B. Singh, T. Gazagnaire, S. Smith, S. Hand, and J. Crowcroft, "Unikernels: Library operating systems for the cloud," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 1, pp. 461–472, 2013.

[78] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted Execution Environment: What It Is, and What It Is Not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[79] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin, "TrustZone Explained: Architectural Features and Use Cases," in *2016 IEEE 2nd International Conference on Collaboration and Internet Computing (CIC).* IEEE, 2016, pp. 445–451.

[80] Z. Ning, F. Zhang, W. Shi, and W. Shi, "Position Paper: Challenges Towards Securing Hardware-assisted Execution Environments," in *Proceedings of the Hardware and Architectural Support for Security and Privacy*, 2017, pp. 1–8.

[81] P. Maene, J. Götzfried, R. De Clercq, T. Müller, F. Freiling, and I. Verbauwhede, "Hardware-Based Trusted Computing Architectures for Isolation and Attestation," *IEEE Transactions on Computers*, vol. 67, no. 3, pp. 361–374, 2017.

[82] INTEL, "Intel software guard extensions," https://www.intel.com/content/www/us/en/architecture-and-technology/software-guard-extensions/supporting-sgx-on-multi-socket-platforms.html, 2021.

[83] "AMD Secure Encrypted Virtualization (SEV)," https://developer.amd.com/sev/, 2021.

[84] O. Demigha and R. Larguet, "Hardware-based solutions for trusted cloud computing," *Computers & Security*, vol. 103, p. 102117, 2021.