vTZ: Virtualizing ARM TrustZone

Zhichao Hua[1,2] Jinyu Gu[1,2] Yubin Xia[1,2] Haibo Chen[1,2] Binyu Zang[1]
Haibing Guan[2]

[1]*Institute of Parallel and Distributed Systems, Shanghai Jiao Tong University*
[2]*Shanghai Key Laboratory of Scalable Computing and Systems, Shanghai Jiao Tong University*
{huazhichao123,gujinyu,xiayubin,haibochen,byzang,hbguan}@sjtu.edu.cn

Abstract

ARM TrustZone, a security extension that provides a secure world, a trusted execution environment (TEE), to run security-sensitive code, has been widely adopted in mobile platforms. With the increasing momentum of ARM64 being adopted in server markets like cloud, it is likely to see TrustZone being adopted as a key pillar for cloud security. Unfortunately, TrustZone is not designed to be virtualizable as there is only one TEE provided by the hardware, which prevents it from being securely shared by multiple virtual machines (VMs). This paper conducts a study on variable approaches to virtualizing TrustZone in virtualized environments and then presents vTZ, a solution that securely provides each guest VM with a virtualized guest TEE using existing hardware. vTZ leverages the idea of separating functionality from protection by maintaining a secure co-running VM to serve as a guest TEE, while using the hardware TrustZone to enforce strong isolation among guest TEEs and the untrusted hypervisor. Specifically, vTZ uses a tiny monitor running within the physical TrustZone that securely interposes and virtualizes memory mapping and world switching. vTZ further leverages a few pieces of protected, self-contained code running in a *Constrained Isolated Execution Environment (CIEE)* to provide secure virtualization and isolation among multiple guest TEEs. We have implemented vTZ on Xen 4.8 on both ARMv7 and ARMv8 development boards. Evaluation using two common TEE-kernels (secure kernel running in TEE) such as seL4 [1] and OP-TEE shows that vTZ provides strong security with small performance overhead.

1 Introduction

ARM TrustZone [20] has been widely used as an approach to providing a TEE for mobile devices including Samsung's Galaxy [14] and Huawei's Mate [17]. TEE has been used to protect security-critical data like cryptographic keys and payment information [45, 42, 54]. Generally, TrustZone provides hardware-based access control of hardware resources, by enabling a processor to run in two asymmetrically-isolated execution environments: a *secure world*, which is a trusted execution environment (TEE), can be configured to access all resources of a *normal world* but not vice versa. To enable TrustZone as a security pillar for ARM-based platform, there have been many secure kernels running in TEEs (called TEE-kernel), including Trustonic [11], Qualcomm's QSEE [7] and Linaro's OP-TEE [6], to host various Trusted Applications (TAs) with different functionalities [21, 55, 43, 42, 38, 66, 59, 8].

While currently TrustZone is mainly deployed on mobile platforms, AMD has integrated TrustZone in their first 64-bit ARM-based SoC solution named "Hierofalcon" [1]. There are also many ARM-based server SoC in the market, including AppliedMicro's "X-Gene 3" [12], Cavium's ThunderX [47], and AMD's "Opteron A1100" [2]. Internet companies like PayPal and Baidu have deployed ARM servers at scale for years [60, 52]. On the other hand, an increasing number of virtualization features have been incorporated into ARM platforms. For example, in ARMv7 architecture, a special CPU mode called *hyp mode* is added for hosting a hypervisor; two-stage address translation together with System Memory Management Unit (SMMU) are also provided to support address translation and secure DMA for virtualization. With such virtualization extensions, commercial virtualization softwares like Xen [37] and KVM [28] have provided built-in support for ARM platform.

With ARM gaining increasing momentum in the server market, one natural question to ask is: *can TrustZone, the security pillar of ARM platform, be leveraged by multiple VMs on a virtualized platform*? Unfortunately, as TrustZone currently provides almost no support for virtualization, all VMs have to share one secure world, which means one TEE-kernel in the secure world serves different TAs for all VMs. Meanwhile, it is known that TEE-kernels are not bug-free and there have been

multiple security vulnerabilities discovered from major TEE-kernel providers including Samsung, Huawei and Qualcomm [61, 62, 24]. Since the TEE-kernel has a higher privilege than any normal world software, once it gets compromised, attackers can access any resources of all VMs as well as the hypervisor, which makes the TEE-kernel the "single point of breach".

One straightforward way to virtualize TrustZone would be using a hypervisor in the normal world to simulate TrustZone without leveraging any features of the hardware TrustZone. However, such an approach heavily relies on the security of the hypervisor, which has a Trusted Computing Base (TCB) with millions of lines of code and usually hundreds of security vulnerabilities discovered [68, 23]. Hence, once a hypervisor is compromised, all guest TEEs (in the following paper, the *guest TEE* presents the virtual secure world for each guest and the *secure world* presents the hardware secure world) are also under attackers' control.

To address these issues, this paper introduces vTZ that provides transparent virtualization of TrustZone while still maintaining strong isolation among guest TEEs with minimal software TCB. The key idea is separating functionality from protection by maintaining one secure co-running VM serving as a guest TEE for each guest, while using the physical TrustZone to enforce strong isolation among them together with the hypervisor. Specifically, vTZ uses two secured modules running within the secure world that interpose memory mapping and world switching. Based on the interposition, vTZ further provides multiple *Constrained Isolated Execution Environments (CIEEs)* that protect self-contained code snippets running inside them by leveraging TrustZone-enabled same-privilege isolation [30, 21, 31] such that the hypervisor cannot tamper with the CIEE or break their execution integrity. The CIEEs are then used to contain the logic that virtualizes the functionalities of the physical TrustZone, including secure booting, secure configuration of memory and devices for each guest TEE. vTZ also provides *Control Flow Locking (CFLock)* to enforce that the CIEEs will be invoked at some specific point and cannot be bypassed. Building atop vTZ, we also provide various VM management operations including VM suspending and resuming while preserving the security properties.

We have implemented vTZ based on Xen-ARM 4.8 on both LeMaker Hikey ARMv8 development board as well as a Samsung's Exynos Cortex development board, and run two common TEE-kernels: seL4 [40, 39] and OP-TEE [6] from STMicroelectronics and Linaro in the guest TEE. The performance evaluation shows that the average applications overhead introduced by vTZ is about 3% compared with Xen.

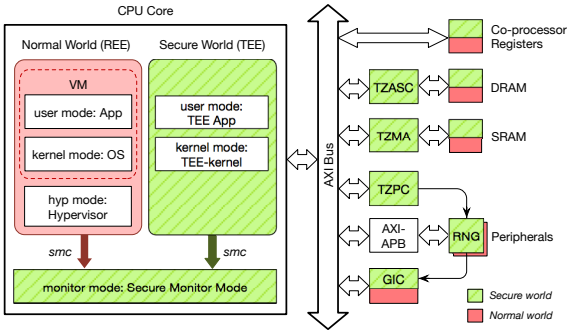


Figure 1: ARM with TrustZone and virtualization extensions: TrustZone splits CPU into normal world and secure world, all other hardware resources are split as well. Each world has its own user and kernel space, and can switch to each other by *smc* instruction. Only the normal world has virtualization support. TrustZone Address Space Controller (*TZASC*): configure DRAM as secure or non-secure (S/NS) partition. TrustZone Memory Adapter (*TZMA*): configure SRAM S/NS partition. TrustZone Protection Controller (*TZPC*): configure peripheral S/NS partition. General Interrupt Controller (*GIC*): control interrupt, can also configure interrupt S/NS partition. Random Number Generator (RNG): device for generating the random number.

2 Background and Motivation

We first give a detail introduction on ARM hardware extensions including TrustZone and virtualization. Then we introduce existing applications of TrustZone and discuss why virtualize it.

2.1 Overview of TrustZone

TrustZone [20] is a hardware security mechanism since ARMv6 architecture, which includes security extensions to ARM System-On-Chip (SoC) covering the processor, memory and peripherals. For processor, TrustZone splits it into two execution environments, a normal world and a secure world (as shown in Figure 1). Both worlds have their own user space and kernel space, together with cache, memory and other resources. It is noted that only the normal world has hyp mode.

The normal world cannot access the secure world's resources while the latter *can* access all the resources. Base on this asymmetrical permission, the normal world is used to run a commodity OS, which provides a *Rich Execution Environment (REE)*. Meanwhile, the secure world, always locates a secure small kernel (TEE-kernel). The two worlds can switch to each other under the strict supervision of a *Secure Monitor* running in monitor mode. Typically, a special instruction called "secure monitor call" (*smc*) is used for worlds switching.

TrustZone divides all memory into two parts: normal part and secure part, which are distributed into normal

world and secure world accordingly. Again, TrustZone ensures that the normal world cannot access the secure part of memory while the secure world can access the entire memory. With this feature, two worlds can communicate with each other by using a piece of shared memory. Besides, the memory partition can be dynamically controlled by the secure world, which gives secure services running in the secure world the ability to dynamically protect certain memory.

For I/O devices and interrupts, TrustZone also splits them into two worlds. An I/O device can be partitioned to one specific world. TrustZone ensures that the normal world cannot access the secure world's I/O devices while the secure world can control the whole system's devices. For each interrupt, TrustZone can designate which world to handle it. When a secure interrupt arrives, TrustZone will switch the processor to the secure world to handle it. Similar to memory, the partitioning of I/O devices and interrupts can be dynamically configured by the secure world.

2.2 Address Translation in ARM

The ARM virtualization extension not only adds a new *hyp mode* in CPU, but brings a complex address translation [4]. ARM architecture leverages translation table, which is pointed by a translation table base register, to perform address translation. There exist two different kinds of address translations: one-stage and two-stage. The one-stage translation simply maps virtual address (VA) to physical address (PA). It is used in the hyp mode (using a hyp-mode translation table) and the secure world (using a stage-1 translation table). Both of them have their own translation table base register. Two-stage translation includes stage-1 and stage-2, which is used by guest VMs. In stage-1, a VA is translated to an intermediate physical address (IPA); in stage-2, the IPA is further translated to the corresponding PA. The stage-1 page table is controlled by guest OS and the stage-2 page table is controlled by the hypervisor.

2.3 TrustZone-based Applications

TrustZone is getting increasing popularity and has been used in various scenarios to protect security-critical data and enhance the security of the normal world.

Secure Storage and Credential Protection: The isolation property of TrustZone makes it an ideal choice to store user's secret data, e.g., private keys, passwords, credit card numbers, etc. For example, a web server could put the private key and all the code accessing it into the secure world [55] so that the private key will never be accessed by the normal world, which can effectively defend against memory exposure attacks such as Heartbleed attack [16] or buffer overread attack [58]. The TrustOTP project [59] provides a secure one-time password (OTP) device on a mobile phone using TrustZone, which can keep working even if the REE OS crashes. Rubinov et al. [54] propose a method to automatically partition a Java application to two parts: one security-sensitive part in secure world and one feature-rich part in normal world. All these systems and technologies can be applied to the server platform in a seamless way.

Enforcing REE Security: TEE can be used to enhance the security of REE because the secure world has higher privilege than the normal world. For example, TZ-RKP [21] provides real-time protection for the normal world kernel from within the secure world. SPROBES [32] provides an introspection mechanism protected by TrustZone that can instrument any instruction of a REE OS, which is able to detect REE kernel rootkit. These technologies can also be used to enforce the security of server OS and applications.

2.4 The Need to Virtualize TrustZone

On mobile phone, the TEE-kernel is usually device dependent and is deployed by the phone manufacturer. For example, Apple [35], Samsung [14] and Huawei [36] use different TEE-kernels in their phones. A phone user is not allowed to install a third party TEE-kernel even after iOS jailbreak or Android root. Each TEE has one root key, which is controlled by the device manufacturer and used for TA authentication. A new TA needs to be signed by the root key before running in the TEE of corresponding device.

Since trustZone is not designed to be virtualizable, currently on a virtualized environment, e.g., cloud, all VMs on the same host have to share one TEE-kernel. Such solution is not only inefficient, but may also cause security issues, since there is no way for each VM to deploy its own TEE-kernel. It means that cloud users are restricted to use the root key controlled by the vender to sign their TAs, and have to trust the only TEE-kernel provided by the vender which is the single point of breach in software running inside TrustZone. Unfortunately, there have been various security vulnerabilities discovered in major vendors' TEE-kernel [62, 61, 63, 64, 15]. These motivate us to design vTZ to virtualize TrustZone to be seamlessly used by multiple virtual machines, while preserving the security properties offered by the hardware-based TrustZone, so that each virtual machine can run its own TEE-kernel in a secure and isolated environment.

3 Design Overview

In this section, we first discuss the goals and challenges of TrustZone virtualization, and present two intuitive de-

Table 1: Properties enforced by TrustZone which should also be enforced by vTZ. Meanwhile, shows how a malicious hypervisor can violate these properties as well as possible results. S/W means secure world. N/W means normal world.

TrustZone Features	System Properties	Properties Violation by Malicious Hypervisor → Consequence
Secure Boot	**P-1.1**: *S/W must boot before N/W.*	Violate boot order. → Secure configuration bypass.
	P-1.2: *Boot image of S/W must be checked.*	Violate integrity check of boot image. → Code injection in guest TEE.
	P-1.3: *S/W cannot be replaced.*	Replace a guest TEE with another one. → Providing malicious TEE.
CPU States Protection	**P-2.1**: *smc must switch to the correct world.*	Switch to a wrong guest TEE. → Providing malicious TEE.
	P-2.2: *Protect the integrity of N/W CPU states during switching.*	Tamper CPU states during switching. → Controlling execution of guest TEE.
	P-2.3: *Protect S/W CPU states.*	Tamper guest TEE's CPU states. → Controlling execution of guest TEE.
Memory Isolation	**P-3.1**: *Only S/W can access secure memory.*	Let arbitrary VM access guest secure memory. → Info leakage.
	P-3.2: *Only S/W can configure memory partition.*	Let arbitrary VM configure guest memory partition. → Reconfigure secure memory as normal.
Peripheral Assignment	**P-4.1**: *Secure interrupts must be injected into S/W.*	Forbid interrupt being injected into guest TEE. → Disturbing the execution of guest TEE.
	P-4.2: *N/W cannot access secure peripherals.*	Let guest N/W access secure peripherals. → Info leakage of secure peripherals.
	P-4.3: *Secure peripherals are trusted for S/W.*	Provide malicious peripherals for guest TEE. → Info leakage of guest TEE.
	P-4.4: *Only S/W can partition interrupt/peripherals.*	Let arbitrary VM configure guest interrupt/peripherals. → Reconfigure secure peripheral as normal.

signs. We then show our threat model, assumptions and the design of vTZ.

3.1 Goals and Challenges

The goal of vTZ is to embrace both strong security as well as high performance. We analyze a set of security properties that physical TrustZone provides, as shown in Table 1, which should be kept after TrustZone virtualization. To enforce these properties, vTZ needs to address the following challenges:

- *Challenge 1*: A compromised hypervisor may violate the booting sequence or booting a compromised or even a malicious guest TEE-kernel.
- *Challenge 2*: A compromised hypervisor may hijack a guest TEE's execution by tampering with its CPU states. It may even switch to a malicious TEE when performing world switching.
- *Challenge 3*: Once the hypervisor is compromised, there is no confidentiality and integrity guarantee for the secure memory owned by a guest TEE.
- *Challenge 4*: Guest TEEs trust peripherals that are configured as secure. A malicious hypervisor may provide a malicious virtual peripheral to a guest TEE.

3.2 Alternative Designs

Design-1: Dual-Hypervisor. This design uses a full-featured hypervisor to virtualize the secure world, which can be called a *TEE hypervisor*, as shown in Figure 2(a). Like the hypervisor in the normal world, the TEE hypervisor is in charge of multiplexing the secure world and offers a virtualized interface of TrustZone to the normal world. It also needs to associate a guest VM with its corresponding guest TEE. However, this design has several issues.

The first issue is its large TCB. A TEE hypervisor needs to virtualize a full-featured execution environment as the physical TrustZone provides, which requires non-trivial implementation complexity due to the lack of virtualization support in the secure world. This leads to a large code base for the TEE hypervisor. Further, since the TEE hypervisor needs to work together with the REE hypervisor to bind one guest TEE for each guest, the TCB of such a design includes not only the TEE hypervisor but also the REE hypervisor. Otherwise, a malicious REE hypervisor may let one guest VM in the normal world switch to another guest's TEE.

The second issue is the poor compatibility with existing TEE-kernels. The TEE-kernel cannot run in the kernel mode, since only the TEE hypervisor can reside at the highest privilege level. While "trap and emulate" may be a viable solution, there are some sensitive but unprivileged instructions, which will silently fail instead of trapping into the hypervisor when being executed in user mode. For example, modifying some privileged bits in *CPSR* register in user mode will just be ignored. Para-virtualization is also possible, but it may lead to compatibility and security problems since existing TEE-kernels have to be modified and the TEE hypervisor has to weaken isolation due to exposing more states and interfaces to guest TEE-kernels.

The third issue is the large overhead due to costly world switching, which involves two hypervisors to trap

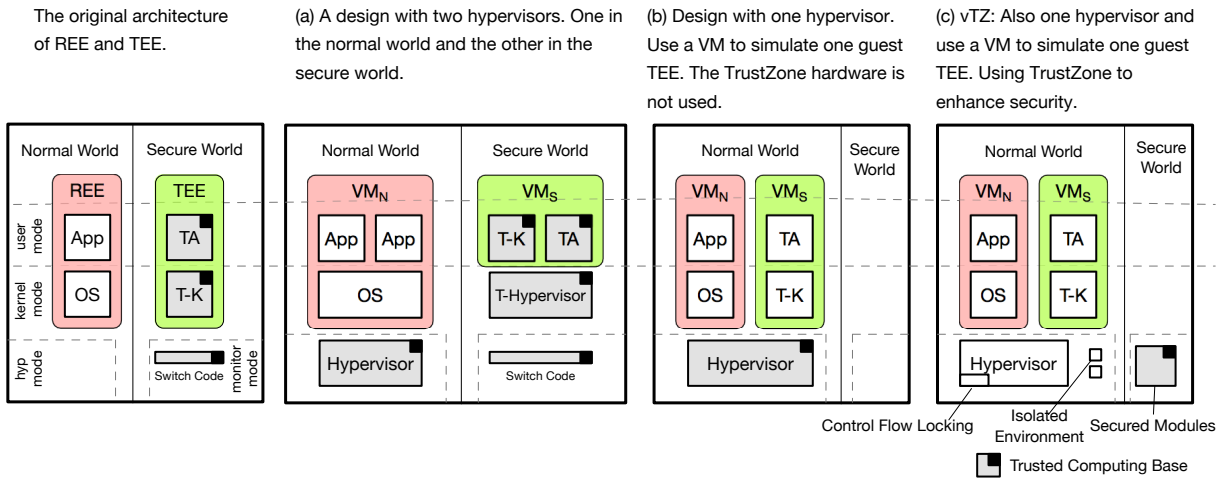


Figure 2: Different designs of TrustZone virtualization. VM_n and VM_s mean the normal world and secure world of a VM, respectively. T-K means TEE-kernel.

and emulates the world switching. An *smc* (secure monitor call) will first be trapped in the REE hypervisor and then transferred to the TEE hypervisor, which in turn transfer the call to the designated guest TEE. This results in a long call path and several privilege level crossings.

Design-2: Full Simulation of TrustZone. This design fully simulates multiple guest TEEs by using the hypervisor running in the normal world to virtualize the guest TEEs along with the normal world guest VMs and to handle their interaction, as shown in Figure 2(b). All switches between two virtualized worlds are simulated by switching between two different VMs (called VM_n and VM_s). This also means that the physical TrustZone is not essential (not used).

Comparing with the first design, this design can achieve better performance, less complexity, and better compatibility. The main problem is its large TCB of a commodity hypervisor, which includes not only the hypervisor but also the management VM (for Xen) or the host OS (for KVM). Either contains millions of lines of code (LoC). There have been 236 and 103 security vulnerabilities uncovered in Xen [19] and KVM [18] respectively, not to mention those in Linux itself.

3.3 Threat Model and Assumptions

vTZ assumes an ARM-based platform that implements the TrustZone extension together with the virtualization extension. All the hardware implementations are correct and trustworthy. vTZ also assumes that the whole system is loaded securely, including both the secure world's and the normal world's code, which is ensured by *secure boot* technology of TrustZone. Intuitively, secure boot only guarantees the integrity of the system during the boot-up process, but not the integrity thereafter. We do not consider side channel attacks or physical attacks to the memory and SoC, like cold boot or bus snooping. We also do not consider the vulnerabilities within a guest TEE.

We assume that any guest VM or guest TEE can be malicious. Like prior work using same privilege level protection [30, 21, 31], we assume the hypervisor itself is not malicious, which is trust during system booting to do initialization correctly. After booting, the hypervisor can be compromised.

3.4 Our Design: vTZ

vTZ adopts the principle of separating functionality from protection [68]. Specifically, vTZ relies on the normal world hypervisor to virtualize functionality of guest TEEs but leverages the physical TrustZone to enforce protection, as shown in Figure 2(c).

To enforce the booting integrity, vTZ uses the secure world to check the booting sequence as well as perform integrity checking (Section 5.1). To provide efficient memory protection, vTZ uses Secured Memory Mapping (SMM), a module in the secure world, to control all the stage-2 translation tables as well as hypervisor's translation table (Section 4.1). Based on that, vTZ can set security policies to, e.g., ensure that any of the guest TEE's

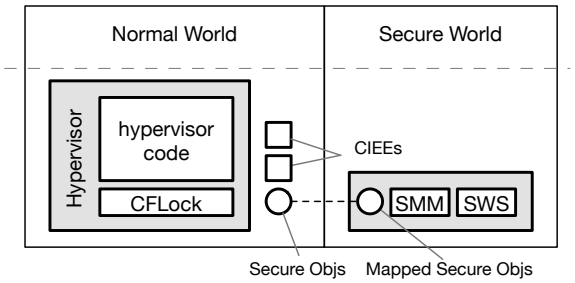


Figure 3: System Architecture: Two secured modules (Secured World Switching (SWS) and Secured Memory Mapping (SMM)) are located in secure world. Constrained Isolated Execution Environment (CIEE) is running in hyp mode in the normal world. Control-flow Lock (CFLock) provides non-bypassable hook for CIEE and secured modules when an exception happens.

The key to achieving the above interposition and protection is efficiency and vTZ achieves these by extending prior work with same privilege isolation [30, 21, 31] to virtualized environments. Specifically, vTZ provides a set of Constrained Isolated Execution Environments (CIEEs) in hyp mode (Section 4.4). CIEE is isolated from each other as well as from the hypervisor. To prevent sophisticated attacks like ROP [53], vTZ ensures the control flow and data flow integrity for code snippets within each CIEE. Control Flow Locking (CFLock) (Section 4.2) is used to provide non-bypassable hook for CIEE and secured modules (SMM and SWS) in the secure world. CIEE is not part of TCB since a compromised CIEE cannot affect the security of other CIEEs or guest TEEs.

The TCB of vTZ contains only the secured modules running in the secure world, which has less than 2k LoC. In contrast, the TCB for design-1 contains the huge REE hypervisor as well as a smaller TEE hypervisor containing tens of thousands of LoC (e.g., NOVA [57] contains a 9k LoC microhypervisor and 29k LoC VMM, OKL4 [33] contains 9.8k LoC). The TCB for design-2 includes a whole commodity hypervisor, which contains several millions of LoC.

4 Protection Mechanisms

As mentioned in Section 3.4, vTZ leverages four mechanisms: SMM, CFLock, SWS and CIEE to enforce the security properties for guest TEEs, as demonstrated in Figure 3. This section will describe all these mechanisms in details.

4.1 SMM: Secured Memory Mapping

The Secured Memory Mapping (SMM) module runs in the secure world for memory isolation. SMM controls the VA-to-PA mapping in the hyp mode as well as IPA-to-PA mappings for guest VMs in an exclusive way. It provides two interfaces to the hypervisor, one for loading a translation table and the other for modifying entries of a translation table. Based on that, SMM manages and enforces different security policies on every memory mapping operation. It only provides one interface to some specific CIEEs which contain virtual partitioning controller emulator (introduced in Section 5.3) to configure security policies.

Exclusive Control of Memory Mapping: To ensure that only the SMM can load or change memory mapping, vTZ enforces that the hypervisor does not contain any instruction to do so. There are three ways for a hypervisor to modify memory mapping: changing the base register to a new translation table [2], changing the entries of translation table, or disabling the address translation [3]. In vTZ, the hypervisor is modified so that is no instruction that loads new translation table or disables translation. Meanwhile, all the page tables are set as read-only to the hypervisor. The corresponding operations are replaced to invocations to SMM.

SMM maps the code pages of the hypervisor as read-only so that the code will not be changed at runtime. It also controls the swapping of hypervisor code and ensures the integrity of code during swapping in process. To prevent return-to-guest attack, SMM ensures that no guest memory will be mapped as executable in hypervisor's space. Moreover, SMM forbids to map any new page as executable in the hypervisor's address space after system booting. vTZ also ensures that there is no ROP gadget that can be used to form new instructions to operate the translation table (which is relatively trivial on ARM platform since instruction alignment is required). We also consider all the ISAs with different length.

4.2 CFLock: Control Flow Locking

Locking the control flow means enforcing a control flow transfer to specific code at some certain event, so that the code will not be bypassed to handle the event. CFLock can "lock" the control flow when any exception happens, which is used to ensure the non-bypassability of the SWS and CIEEs (described in following sections). We force the control flow at the entry of different exception handlers. The ARM architecture uses a special register to point to the base address of exception table, and each table entry will correspond to one exception handler. We deprive the hypervisor of ability to modify this base register by replacing all the modification instructions with invocations to secure world, similar as Section 4.1. The exception table will also be marked as read-only by SMM, to enforce that each exception will eventually reach to one specific handler. After that, certain control flow transfer instructions (e.g., an uncondi-

tional jump to the entry of a CIEE) will be implanted in these handlers. Since SMM ensures the code of hypervisor is read-only, such instruction will never be modified.

4.3 SWS: Secured World Switching

Secured World Switching (SWS) is a module running in the secure world enforcing the security properties of guest's world switching. SWS interposes two types of switching: between one guest's VM_n and VM_s, as well as between a virtual machine and the hypervisor. In vTZ, these two types can be handled uniformly since the former is also handled by the hypervisor. To achieve complete interposition, SWS ensures that each switching will first trap to SWS itself.

Complete Interposition: On ARM architecture, there are two situations causing switching from a guest VM to the hypervisor. One is *IRQ/FIQ interrupt* which is caused by hardware interrupts, the other is *guest sync exception* (hyp trap in ARM32), which is caused by any trapping instructions (like *smc*, co-processor accessing, hypercall, etc.) or data abort (like stage-2 translation table abort). In both situations, switching is caused by exception. Hence, CFLock can be used to enforce that the control flow will eventually be transferred to SWS.

If the hypervisor needs to switch to a guest, it must change the current exception level from the hyp mode to kernel mode. There are three methods for the hypervisor to do so: by executing *eret* instruction, by executing *movs pc, lr* instruction, or directly setting the exception level (e.g., by executing *CPS* instruction); In order to enforce a single exit point, SWS requires the hypervisor to remove all these instructions and replace them with corresponding invocations to SWS. Thus, SWS ensures that its interposition will be non-bypassable for each control transfer between guest VMs and the hypervisor.

4.4 CIEE: Constrained Isolated Execution Environment

CIEE is an environment in the hyp mode, which is used to implement some key logics that emulate the functionalities of TrustZone, e.g., virtualizing partition controller (see Section 5.3). Each CIEE has its own translation table, stack page and secure objects, and contains a piece of code with a strong control flow integrity as well as data flow integrity. Meanwhile, SMM and SWS will assign different capabilities to different CIEEs, so that one CIEE can only access its own secure objects. Each CIEE has different copies of secure objects for different guests, and one CIEE can only serve one caller guest at a time.

Enforce Security of CIEE: In order to protect CIEE from a compromised hypervisor and ensure that it cannot be bypassed, it must satisfy following requirements:

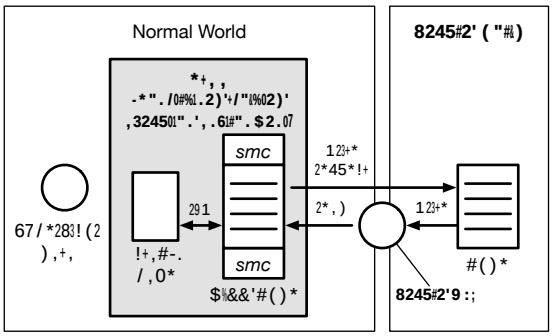


Figure 4: Constrained Isolated Execution Environment (CIEE): an execution environment which is fixed on memory location to identify itself to the secured modules running in the secure world. Its execution only depends on the stack page and secure objects. Interrupts are disabled when in CIEE.

1. **Single entry point**: It is illegal to jump to the middle of a CIEE.
2. **Run-to-completion**: Once starting to run, it must run to the completion without being interrupted.
3. **No dependence on the hypervisor's data**: Otherwise a compromised hypervisor can affect CIEE's execution.
4. **No data exposure to the hypervisor**: Otherwise a compromised hypervisor may tamper with CIEE's running states.
5. **Unforgeable to the secure world**: A CIEE needs to identify itself to the modules running in the secure world; otherwise the hypervisor may impersonate to be CIEE.

Figure 4 shows the design of *CIEE* in vTZ. Each CIEE's code is loaded to a fixed memory address during initialization. The secure world is aware of CIEE's metadata, which includes $\{entry\_addr, exit\_addr\}$, to meet the requirement-⑤. The code pages of CIEE are mapped as read-only by SMM to ensure the code integrity. By default, the code pages of CIEE are mapped as non-executable. The first and last instructions of a CIEE are always *smc*. After the first *smc* (on a different executable page) trapping to the secure world, vTZ will check the trapping address stored in *LR_mon* [4] and to identify the specific CIEE by the address. Then SMM will remap the corresponding CIEE's code body to be executable, and map stack pages as read/write. The remapping is done only on the current CPU core; thus on other cores the CIEE is still non-executable and its stack is not accessible. In this way, the execution must start from the first *smc*, and the requirement-① is satisfied.

vTZ will disable any interrupt before transferring control to a CIEE. The code in CIEE must be self-contained,

and will keep running till the last *smc* instruction. Thus, the requirement-② is met. When trapping due to the last *smc*, SMM will again remap the code body to non-executable and clean the stack page to meet requirement-④. There is no need for the hypervisor to access CIEE's code, secure objects or CIEE's stack page.

To further enforce the control flow integrity, protecting the code page alone is not enough. We also ensure that CIEE's code only depends on either local data in stack page or secure objects. The former is only mapped when CIEE is executed, and the latter is mapped as read-only in the hyp mode and can only be modified by invoking the secured modules. So that we can satisfy requirement-③, and hypervisor cannot tamper with CIEE's control flow.

Since a CIEE contains code which serves the guest or the hypervisor, vTZ enables a CIEE to write results back to them. The memory used to store results for a guest is marked as secure object, and the CIEE needs to ask the SMM to write it. The hypervisor's data is mapped to the CIEE, so that the CIEE can directly write results to the hypervisor's memory. For example, the CIEE for VM suspending (Section 5.4) could directly return the encrypted snapshot of a VM_s back to the hypervisor.

Privilege Isolation: CIEE is not in the TCB of vTZ. vTZ isolates the privilege of different CIEEs from two dimensions. First, one CIEE cannot access any sensitive data not belonging to it. This is enforced by SMM which ensures one CIEE's own secure object and stack page will never be mapped into other's address space. Second, one CIEE can only provide service to the current guest. One CIEE will have different secure objects for different guests. SWS can identify the current guest and then only allows a CIEE to access current guest's secure objects.

5 TrustZone Virtualization

In this section, we present how to virtualize TrustZone features listed in Table 1 by using the four mechanisms described in last section. We also demonstrate the process of suspending and resuming a guest with its guest TEE.

5.1 Virtualizing Secure Boot

Secure boot is used to ensure the integrity of booting. The booting process of a TrustZone-enabled device includes following steps: 1) Loading a bootloader from ROM, which is tamper resistant. 2) The bootloader initializes the secure world and loads a TEE-kernel to memory. 3) The TEE-kernel does a full initialization of the secure world, including the secure world translation table, vector table and so on. 4) The TEE-kernel switches to the normal world and executes a kernel-loader. 5) The kernel-loader loads a non-secure OS and runs it.

During the process, each time a loader loads a binary image, it will calculate the checksum of the image to verify its integrity. Meanwhile, the booting order is also fixed: the TEE-kernel is the first to run so that it can initialize the platform first. To virtualize the secure boot process, vTZ is required to enforce the following properties:

- **P-1.1**: *S/W must boot before N/W.*
- **P-1.2**: *Boot image of S/W must be checked.*
- **P-1.3**: *S/W cannot be replaced.*

The virtualized secure boot process of vTZ is shown in the top part of Figure 5. The hypervisor initializes the data structure and allocates memory for both guest VM_n and its corresponding VM_s, loads the TEE-kernel image and guest normal world OS image to the memory, respectively. Then the hypervisor will register the two VMs in the Secured World Switching (SWS) module. Since SWS controls all the world switches between the *hyp mode* and a VM, it can ensure that only registered VM can be executed. During registration, SWS first asks the Secured Memory Mapping (SMM) module to remove all the mapping of memory pages allocated to the guest from the hypervisor's translation table and checks the integrity of a guest's TEE-kernel. Then SWS creates a binding between the guest VM_n and VM_s by recording their VMID, and marks their context data as read-only to hypervisor. SMM will also initialize the stage-2 translation tables of these two VMs and set the VMID in the stage-2 translation table base register. So the **P-1.2** and **P-1.3** are enforced. The scheduling of VM is done by the hypervisor. SWS will ensure that the VM_s must run before the corresponding VM_n to enforce **P-1.1**.

5.2 Protecting CPU states

vTZ needs to enforce following properties to provide the same CPU states protection of TrustZone.

- **P-2.1**: *smc must switch to the correct world.*
- **P-2.2**: *Protect the integrity of N/W CPU states during switching.*
- **P-2.3**: *Protect S/W CPU states.*

SWS intercepts all the switching between a guest VM and the hypervisor, A switching includes saving states of the current VM, finding the next VM, and restoring its states. The states saving and restoring are done by SWS in the secure world, while the finding of next VM is done by the hypervisor, as shown in the bottom half of Figure 5. Then SWS can check the restored target VM

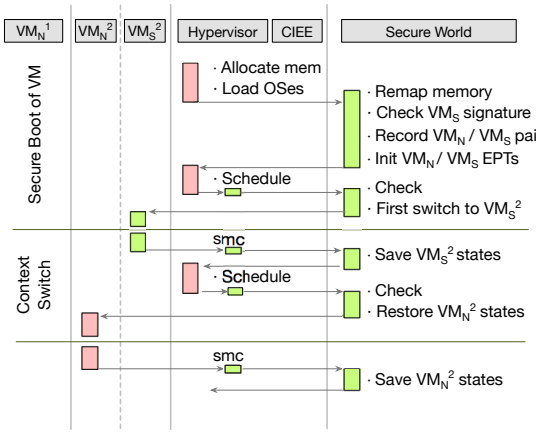


Figure 5: Boot and context switch process: Hypervisor is responsible to build VM for each guest. SWS verifies every guest before hypervisor can execute it, including initializing guest EPT and checking image integrity. Let hypervisor switch between VM_s and VM_n. SWS checks all entering to a VM.

to ensure **P-2.1** and **P-2.2** are satisfied. During execution, SWS also prevents the hypervisor from stealing or tampering with VM_s's context to achieve **P-2.3**. For example, if one VM is exited because of the scheduling, then its CPU states cannot be modified. Further, VM_s's system control registers also cannot be modified by the normal world hypervisor.

5.3 Virtualizing Resource Partitioning

TrustZone can split hardware resources to the normal world or the secure world. Three different resource partitions are provided, together with three different controllers which are used to configure the partition:

- **Memory partitioning**, which is configured by TrustZone Address Space Controller (*TZASC*).
- **Peripheral partitioning**, which is configured by TrustZone Protection Controller (*TZPC*).
- **Interrupt partitioning**, which is configured by General Interrupt Controller (*GIC*).

Once set as secure, the resource can only be accessed by the secure world. A secure interrupt must be injected to the secure world and will lead to a world switching if it happens in the normal world. All the three controllers can be used to repartition the resource only by the secure world.

It is needed for a guest to dynamically partition some critical virtual devices such as virtual Random Number Generator (vRNG), vGIC and so on. For example, a guest may only allow its guest TEE to configure the vGIC, or first initialize a vRNG in VM_s and then use it in VM_n. To support these requirements, vTZ provides the same semantic of resource partitioning as a real TrustZone, which includes the configuration of partitioning and the enforcement of partitioning.

Virtualizing Partitioning Controllers: Following two properties should be satisfied:

- **P-3.2**: *Only S/W can configure memory partition.*
- **P-4.4**: *Only S/W can partition interrupt/peripherals.*

The virtualization of partitioning configuration is done by the classic "trap and emulate" method. vTZ provides three virtual controllers (*vTZASC*, *vTZPC* and *vGIC*) for each guest. ARM only provides memory mapped I/O, all devices are operated by accessing their device memory region. By mapping all the three controllers' memory regions as read-only in each guest's stage-2 translation table, any write to them will cause a trap to the hypervisor. The CFLock enforces that the trap will be handled by a handler in a CIEE. The handler first checks whether the trap is raised by a VM_s, and will ignore it if not, which enforce **P-3.2** and **P-4.4**. The handler then invokes a corresponding controller emulator to do the configuration. The emulator runs in another CIEE and can only repartition for the guest who performs the write operation.

Secure Memory Partitioning: The following property is a fundamental one:

- **P-3.1**: *Only S/W can access secure memory.*

This property is enforced by the SMM module through the following policy: any guest's secure memory region can only be mapped in its VM_s but not any other VMs or the hypervisor.

Secure Device Partitioning: In secure device part, vTZ must enforce the following two properties:

- **P-4.2**: *N/W cannot access secure peripherals.*
- **P-4.3**: *Secure peripherals are trusted for S/W.*

We have implemented commonly used secure devices for existing TEE-kernel, like *TZASC*, *TZPC*, *GIC*, *uart*, *RTIC* and so on. We virtualize these devices for each guest by "trap and emulate". Since all ARM devices use memory mapped I/O, access to a virtual secure device can be easily trapped by controlling stage-2 translation table. When a virtual secure device is accessed, the CPU will get trapped. The trap will be handled by a corresponding emulator, which runs

that it cannot be accessed by this guest's VM_n, so the **P-4.2** is enforced. One guest's configuration will not influence others. The entire process is also protected by CFLock, so that all the operations on virtual secure device will eventually be handled by the emulator in CIEE, and **P-4.3** is satisfied.

Secure Interrupt Partitioning: For interrupts, vTZ needs to ensure the following property:

- **P-4.1**: *Secure interrupts must be injected into S/W.*

Each time when an interrupt happens, it will be handled by a secure interrupt dispatcher. The dispatcher can decide whether the interrupt is secure or not according to a virtual interrupt partition list which is managed by trusted virtual GIC. The CFLock enforces that the dispatcher cannot be bypassed and the CIEE ensures the security of it. These work together to enforce **P-4.1**.

5.4 Supporting TEE Management Operations

Suspending and resuming are two important operations of virtualization. vTZ enforces the correctness and security of these operations, as shown in Figure 6.

Guest TEE Suspension/Resumption: The hypervisor needs to ask a suspend CIEE to save all secure states of one guest. The suspend CIEE will first invoke the SWS to encrypt and hash guest's vTZ related data, including CPU states, memory partition, interrupt partition, device partition, etc. Then it asks the SMM to encrypt and hash all guest's secure pages and share them with the hypervisor. The hypervisor stores all these encrypted data and hash value together with guest VM_n's states into a snapshot file. Resuming process is similar with that for suspending but in the opposite direction.

6 Performance Evaluation

We evaluate the performance of vTZ on both a Hikey ARMv8 development board (64-bit) and an Exynos Cortex ARMv7 development board (32-bit). The Hikey board enables eight 1.2 GHz cores together with 2GB memory. The Exynos board enables one 1.7 GHz core and 1GB memory. We use Xen 4.4 [13] as the hypervisor and Linux 4.1 as the guest normal world kernel and Dom0 kernel. For guest TEE-kernel, We ported two widely used TEE-kernels, namely seL4 [9] and OP-TEE [6], to vTZ. On the Exynos board, each guest together with Dom0 has one virtual CPU. On the Hikey board, each guest, as well as Dom0, has one virtual CPU and each virtual CPU is pinned on one physical CPU. We leverage ARM's performance monitor unit (PMU) to measure the clock cycles.

Running Existing TEE-kernel: Running a TEE-kernel on vTZ needs three steps. First, vTZ leverages Xen's multi-boot loader to load TEE-kernel's image, so we need to add a multi-boot header in the image. Second, we add a new description file (e.g., platform_config.h in OP-TEE) to describe the memory layout of our guest TEE. Finally, since vTZ already provides a secure context switch, we remove the context switching logic in TEE-kernel.

6.1 Micro-benchmark

World Switch Overhead: For the physical TrustZone, the time of switching between two worlds is about 17,840 cycles on Exynos board and 1,294 cycles on Hikey board. The cost includes context saving and restoring in the monitor mode (shown in Table 2). In vTZ, one switching between guest's normal world and guest TEE is about 34,164 cycles on Exynos and 6,837 cycles on Hikey. The overhead is still acceptable since world switching happens rarely and thus has little effect on TrustZone-based applications.

Secure Configuration Overhead: A TEE-kernel usually configures system resource partitioning during initialization or occasional run-time protection. Table 2 shows the overhead of these configurations in vTZ. The native value is performing configuration by hardware in the real secure world. Since HiSilicon, the vender of hikey's SoC, does not publish the register mapping of *TZASC* or *TZPC*, the native time of Hikey is not provided. Same as world switching, secure configuration operations happen rarely, so the overhead will have limited effect on the whole system.

Run-Time Integrity Checker: Besides virtualizing secure devices like *vTZASC*, *vTZPC* and *virtual GIC* (virtual devices used to perform resource partition), we also virtualize and use Run-Time Integrity Checker (RTIC) to evaluate the overhead of vTZ's virtual secure devices. *RTIC* is a commonly used security-related device which can calculate hash values of at most five different memory regions. TEE-kernel can leverage it to detect whether some memory regions have been tampered with. We leverage RTIC to perform SHA1 hashing on five memory regions with sizes from 1K to 128K. Figure 7(a) shows the overhead of *vRTIC* (virtual RTIC) emulated by vTZ in CIEE. It only incurs overhead from 0.3% to 4%.

Table 2: Single operation overhead (unit: *cycle*). Native means real TrustZone and vTZ is our system.

	Native (ARMv7)	**vTZ** (ARMv7)	**Native** (ARMv8)	**vTZ** (ARMv8)
World Switching	17840	34164	1294	6837
Memory Partition	5798	10341	n/a	7918
Device Partition	1886	10395	n/a	7289
Interrupt Partition	1073	4031	755	2903

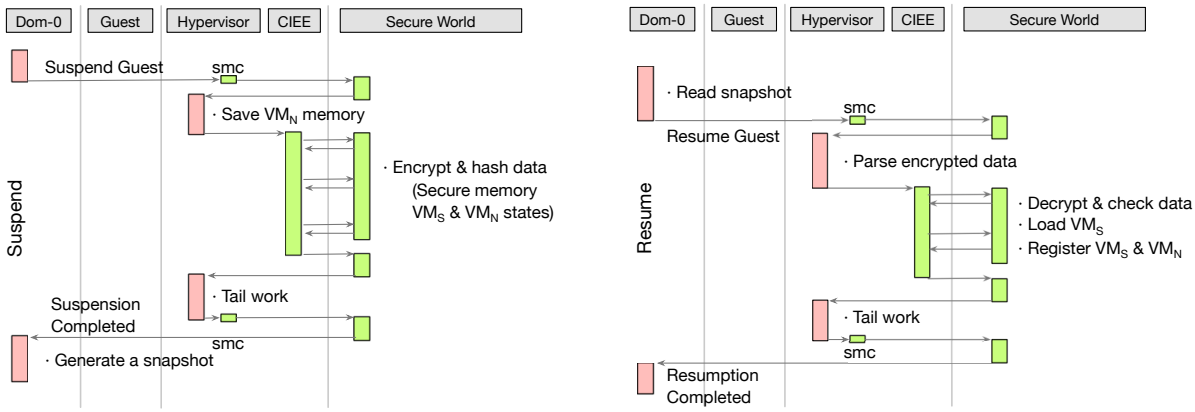


Figure 6: TEE suspension and resumption: During suspension operation, all VM_s's memory and CPU states are encrypted in secure world, and a hash value is computed as well. During resumption operation, such states are decrypted in secure world and SWS will then verify the integrity.

6.2 Application Overhead

Single Guest: We test four real applications (ccrypt, mcrypt, GnuPG and GoHttp) and compare them with original Xen on ARMv7 and ARMv8 platform. We use these applications to encrypt/transfer file about 1KB, and protect the encryption logic in real secure world/guest TEE. The application/guest VM is pinned on one physical core in native environment/virtualization environment, respectively. Figure 7(b) and Figure 7(c) show the overhead. Our system has little overhead compared with original Xen on both ARMv7 and ARMv8 platform.

Multi-Guest: We compare the concurrent performance of vTZ with native environment, real TrustZone and original virtualization environment (Xen). The GoHttp server is used to do the evaluation, and we protect its encryption logic in secure world/guest TEE. In native environment (including protection with TrustZone), each GoHttp server runs as a normal process. In virtualization environment (including protected by vTZ), each of them runs in one guest VM. The client, which sends the https request, runs in the same guest with the server to bypass the network overhead. Each client downloads a 20M file from the server. We do not evaluate more applications concurrently because the memory on the board limits the number of VMs. The results are shown in Figure 8.

On ARMv7 platform, which only has one core enabled, the virtualization (Xen hypervisor) itself brings non-negligible (about 40%) performance slowdown. While on ARMv8 platform, the overhead is remitted with the benefit of 8 enabled cores. The overhead of virtualization becomes larger compared with the single case because that the GoHttp server transfers a big file (20 MB) for each request. Such transferring time is larger than Xen scheduler's time slices (30ms), and then the scheduler will influence the performance. Finally, vTZ has about a 5% performance slowdown compared with original Xen on ARMv8 implementation, and less than 30% performance slowdown compared with native environment.

6.3 Server Application Overhead

We also evaluate two widely used server applications, MongoDB [5] and Apache [3] on Hikey board. Same as the multi-guest evaluation, we run applications on four different environments. Meanwhile, the clients are executed together with the server to bypass the network overhead. One difference is that the guest has eight virtual cores instead of one.

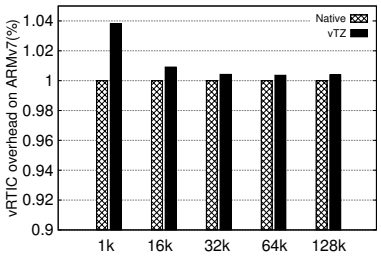
Figure 9(a) shows the insert operation throughput of MongoDB. The client continually inserts object to the server. We evaluate the throughput with different sizes of objects. The result shows that vTZ has little overhead compared with virtualization environment. For Apache (shown in Figure 9(b)), we evaluate the downloading throughput by downloading a file (size is 100MB) from the server with https protocol. The result shows that using virtual TrustZone caused less than 5% overhead in virtualization environment.

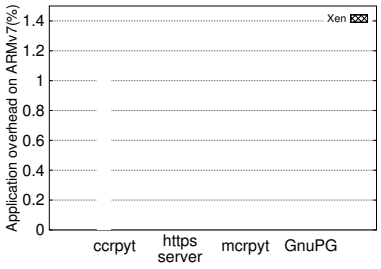
7 Security Analysis

In this section, we assume a strong adversary who can directly boot a malicious guest (including a malicious guest TEE) and even compromise the hypervisor.

7.1 Breaking TrustZone Properties

Booting Protection: A compromised hypervisor can tamper with the system image loaded in guest TEE. The Secured World Switching (SWS) module will check the integrity of the image before execution, and only allow the hypervisor to enter a registered guest TEE. Meanwhile, the attacker may want to boot a guest VM_n before


vRTIC overhead on ARMv7(%)
1.04
1.02
1
0.98
0.96
0.94
0.92
0.9
Native
vTZ
1k
16k
32k
64k
128k


Application overhead on ARMv7(%)
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Xen
ccrpyt
https server
mcrpyt
GnuPG

(a)

visor, CIEEs or even the secured modules in real secure world. The secure boot technology provided by hardware enables vTZ to ensure the integrity of all secured modules, hypervisor and all CIEEs during system boot. After that, the SMM will ensure that all codes in hyp mode are write-protected. Meanwhile, the SMM never allows mapping any new executable memory in the hyp mode after system boot to forbid code injection into the hyp mode. A benign hypervisor also does not need to load code dynamically.

Code-reuse or Return-to-guest Attacks: Attacker may try to reuse code of the hypervisor or let the hypervisor jump to some code region of a guest VM to execute critical instructions (e.g., switching translation table) and bypass the SMM. ARM has several ISAs (e.g., aarch64, aarch32), the instructions of them are fix-byte aligned. vTZ ensures there is no key instruction under any ISAs in the hyp mode's text section, so that there is no ROP gadget in the code to reuse. Meanwhile, the SMM ensures that only the code of the hypervisor can be mapped as executable in the hyp mode, thus return-to-guest attack also can be prevented.

DMA Attack: An attacker may try to access guest secure memory or inject code into hypervisor's memory by leveraging Direct Memory Access (DMA). vTZ defends this attack by controlling System Memory Management Unit (SMMU), which performs address translation for DMA. SMMU is controlled by certain memory mapped registers. It is ensured that these memory regions are only mapped in the secure world. After exclusively controlling the SMMU, we can ensure that all DMAs cannot access guest's secure memory, hypervisor's text section or CIEE's memory.

Debugging Attack: The attacker may also want to bypass the SWS or CFLock by setting debug checkpoint on the *smc* instruction. Then she can perform some operations before switching to SWS or CFLock works. vTZ controls the entry points of all limited exception handlers, and the debug procedure is also under control. Thus, the debug point on *smc* instruction in the hyp mode will trigger an infinite iteration, since the first instruction a debug exception handler executed is also *smc*. This is a kind of DoS attack and is not considered in this paper.

Security of CIEEs: While CIEE contains some logic which provides critical services for guest TEE, e.g., virtualizing TZPC (TrustZone Protection Controller), vTZ still excludes it from the system TCB. Although there is work which can verify some small piece of privileged code (e.g., Jitk [65]), vTZ currently does not formally verify the code in CIEE. Hence, even all CIEEs are small and virtuous, they may still contain bugs. For example, the attacker first compromises the CIEE responsible for virtualizing memory configuration device. Then she tries to configure other guest's memory partition or compromise other CIEEs. vTZ prevents these attacks by constraining a CIEE's abilities. First, SWS can identify current guest TEE and SMM will forbid a CIEE to access any data belonging to other guests. Second, SMM ensures that different CIEEs have different memory mapping and can only access their own secure objects.

7.3 Security Limitations

As described in the threat model, vTZ does not consider hardware-based attacks, e.g., the cold-boot attack, or side-channel attacks. Meanwhile, vTZ can not prevent DoS attacks, e.g., a hypervisor may never execute a guest TEE. But note that vTZ can ensure that if a guest VM uses *smc* instruction to switch to its guest TEE, such switching cannot be ignored.

TEE-kernels may have bugs [61, 62, 24]. Such bugs enable an attacker to directly get data from a guest TEE or even to get control of it. Defending against these attacks is not the goal of vTZ, as the real TrustZone also cannot handle it. However, vTZ does ensure that a compromised TEE-kernel can only affect its corresponding guest while cannot bypass the isolation or attack other TEE-kernels. In all, vTZ aims to achieve the same security level with TrustZone, but no more.

8 Related Work

ARM-based servers are increasingly getting more attentions [50, 51, 25]. It can be expected that virtualization, as one of the most important enabling technologies in the cloud, will also be prevalent on ARM-based servers. There have already been many TrustZone based systems [55, 59, 54, 21, 32, 14], as mentioned in Section 2.3. Many of these designs are not specific for mobile devices, which can be used on ARM servers. vTZ paves the way for the use of TrustZone guarantees in similar environments.

Virtualization of Security Hardware: Similar as TrustZone, Trusted Platform Module (TPM) is a hardware extension for security. vTPM [48] makes TPM functions available to virtual machines by virtualizing the TPM hardware to multiple virtual vTPM instances, which supports suspending and resuming operations. vTZ shares a similar goal with vTPM but is harder since the interaction between the secure world and the normal world is much more complex than TPM. fTPM [49] presents a design and implementation of a TrustZone-based TPM-2.0, which is a pure software solution without the need of a real TPM chip. It can be seen as an application of TrustZone, and by using vTZ fTPM can now inherently support virtualization as well.

Other Hardware-based TEE: Intel's Software Guard eXtension (SGX) [10] offers a strongly isolated execu-

tion environment that can defend against physical attacks. AMD has also announced two security features [34] named Secure Memory Encryption (SME) for defending against physical attack and Secure Encrypted Virtualization (SEV) for protecting VM against hypervisor. Unlike TrustZone, an SGX-TEE does not have higher privilege and can only run in user mode, which makes it not suitable for certain scenarios like security monitoring. Meanwhile, these technologies focus more on memory isolation while TrustZone can also support peripheral partitioning (e.g., random number generator, trust timer, secure co-processors, etc.) Further, currently on ARM platform there is no extension like SGX or SME/SEV while TrustZone has already been widely deployed and has various applications.

Software-based TEE: There are many types of TEE that are based on hypervisor [68, 56, 26, 67, 44, 41, 46], or based on Linux kernel [31, 22, 27], or based on compiler [30, 29], to name a few. These researches are orthogonal to our work. vTZ does not try to provide a new abstraction of TEE but aims to virtualize the existing TrustZone hardware and let all the guests have their own TEE without trusting a large TCB.

9 Hardware Design Discussion

In this section we discuss how to modify TrustZone hardware to support virtualization. One design choice is adding virtualization extension (e.g., hyp mode) in the secure world. After that, software developers can run a TEE hypervisor inside the secure world to virtualize multiple TEEs for different guests (similar with Design-1 in Section 3.2). Although this design simplifies the implementation of the TEE hypervisor, the whole system's security still depends on the interaction between two hypervisors.

Another design is to make the hypervisor unware of the TrustZone. When a virtual machine executes a *smc* instruction, it will directly switch from the normal world to secure world or vice versa, without trapping to the hypervisor. The CPU states are protected by the hardware as before. Both the secure world and the normal world share the same guest physical address space, so that the secure world can still access all the memory of its VM, but cannot access other VM's memory. All the hardware resource partition devices (e.g., TZASC) are virtualized (e.g., by trap-and-emulate) for multiplexing. For example, a vTZASC can only be configured by a VM when the VM is running in its secure world. In this design, only one hypervisor is needed. The secure world and normal world runs as a single VM, which also simplifies the scheduling of VCPU.

10 Conclusion

This paper described vTZ, a design aiming at virtualizing TrustZone in ARM architecture. vTZ provides each guest with an isolated guest TEE, which has the same functionalities and security with the physical secure world. vTZ uses a few modules running in the secure world to securely interpose memory mapping, world switching and device accesses. vTZ further leverages Constrained Isolated Execution Environments (CIEEs) in the normal world to virtualize the functionality of TrustZone. vTZ's TCB only contains the secured modules in the secure world together with system's bootloader. The performance overhead incurred by vTZ is shown to be small.

11 Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work is supported in part by National Key Research and Development Program of China (No. 2016YFB1000104), China National Natural Science Foundation (No. 61303011 and 61572314), a research grant from Huawei Technologies, Inc., National Top-notch Youth Talents Program of China, Zhangjiang Hi-Tech program (No. 201501- YP-B108-012), a foundation for the Author of National Excellent Doctoral Dissertation of PR China (TS0220103006), Singapore NRF (CREATE E2S2).

References

[1] Amd launching "hierofalcon" 64bit arm embedded chips in 1h 2015 - zen and k12 next year. http://wccftech.com/amd-launching-arm-serves-year-wip/#ixzz3Yef58mtq.

[2] Amd opteron a1100. `http://www.amd.com/en-gb/products/server/opteron-a-series`.

[3] Apache http server. `https://www.apache.org/`.

[4] Armv8 white paper. `https://community.arm.com/docs/DOC-10896`.

[5] Mongodb. `https://www.mongodb.com/`.

[6] Optee. `https://github.com/OP-TEE/`.

[7] Qualcomm security. https://www.qualcomm.com/products/ snapdragon/security.

[8] Qualcomm smart protect. `https://www.qualcomm.com/products/snapdragon/security/smart-protect`.

[9] sel4. `http://sel4.systems`.

[10] Software guard extensions programming reference. `https://software.intel.com/site/default/files/329298-001.pdf`.

[11] Trustonic inc. `https://www.trustonic.com/`.

[12] X-gene3. https://www.apm.com/news/appliedmicro-announces-x-gene-3-the-industrys-first-armv8-a-fimnfet-server-s/.

[13] Xen on arm. `https://wiki.xen.org/wiki/Xen_ARM_with_Virtualization_Extensions`.

[14] Samsung knox. http://www.samsung.com/global/business/mobile/solution/security/samsung-knox, 2013.

[15] Bugs in htc's tee. http://atredispartners.blogspot.com/2014/08/here-be-dragons-vulnerabilities-in.html, 2014.

[16] Heartbleed. https://en.wikipedia.org/wiki/Heartbleed, 2014.

[17] Huawei. www.vmall.com/product/1372.html, 2014.

[18] Cves in kvm. http://web.nvd.nist.gov/view/vuln/search, 2016.

[19] Cves in xen. http://web.nvd.nist.gov/view/vuln/search, 2016.

[20] ALVES, T., AND FELTON, D. Trustzone: Integrated hardware and software security. *ARM white paper 3*, 4 (2004), 18–24.

[21] AZAB, A. M., NING, P., SHAH, J., CHEN, Q., BHUTKAR, R., GANESH, G., MA, J., AND SHEN, W. Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (2014), ACM, pp. 90–102.

[22] AZAB, A. M., SWIDOWSKI, K., BHUTKAR, J. M., SHEN, W., WANG, R., AND NING, P. Skee: A lightweight secure kernel-level execution environment for arm. In *Network & Distributed System Security Symposium (NDSS)* (2016).

[23] BAUMANN, A., PEINADO, M., AND HUNT, G. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS) 33*, 3 (2015), 8.

[24] BEHRANG. A software level analysis of trustzone os and trustlets in samsung galaxy phone. https://www.sensepost.com/blog/2013/a-software-level-analysis-of-trustzone-os-and-trustlets-in-samsung-galaxy-phone/, 2013.

[25] BOGGS, D., BROWN, G., TUCK, N., AND VENKATRAMAN, K. Denver: Nvidia's first 64-bit arm processor. *IEEE Micro 35*, 2 (2015), 46–55.

[26] CHEN, X., GARFINKEL, T., LEWIS, E., SUBRAHMANYAM, P., WALDSPURGER, C., BONEH, D., DWOSKIN, J., AND PORTS, D. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *Proc. ASPLOS* (2008), ACM, pp. 2–13.

[27] CHEN, Y., REYMONDJOHNSON, S., SUN, Z., AND LU, L. Shreds: Fine-grained execution units with private memory. In *Security and Privacy (SP), 2016 IEEE Symposium on* (2016), IEEE, pp. 56–71.

[28] CHRISTOFFER DALL, J. N. Kvm/arm:the design and implementation of linux arm hypervisor. In *ASPLOS* (2014).

[29] CRISWELL, J., DAUTENHAHN, N., AND ADVE, V. Virtual ghost: Protecting applications from hostile operating systems. *ACM SIGARCH Computer Architecture News 42*, 1 (2014), 81–96.

[30] CRISWELL, J., GEOFFRAY, N., AND ADVE, V. S. Memory safety for low-level software/hardware interactions. In *USENIX Security Symposium* (2009), pp. 83–100.

[31] DAUTENHAHN, N., KASAMPALIS, T., DIETZ, W., CRISWELL, J., AND ADVE, V. Nested kernel: An operating system architecture for intra-kernel privilege separation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (2015), ACM, pp. 191–206.

[32] GE, X., VIJAYAKUMAR, H., AND JAEGER, T. Sprobes: Enforcing kernel code integrity on the trustzone architecture. *arXiv preprint arXiv:1410.7747* (2014).

[33] HEISER, G., AND LESLIE, B. The okl4 microvisor: convergence point of microkernels and hypervisors. In *Proceedings of the first ACM asia-pacific workshop on Workshop on systems* (2010), ACM, pp. 19–24.

[34] INC., A. Amd memory encryption, white paper. http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf.

[35] INC., A. ios security guide. https://www.apple.com/business/docs/iOS_Security_Guide.pdf, 2016.

[36] INC., H. Built-in tee chip for enhanced security for your private data. http://phoneproscons.com/716/huawei-honor-magic/352/built-in-tee-chip-for-enhanced-security-for-your-private-data/, 2016.

[37] J, HWANG, S., SUH, S., HEO, C., PARK, J., RYU, S., PARK, C., AND KIM. Xen on arm: System virtualization using xen hypervisor for arm-based secure mobile phones. In *IEEE CCNC* (2008).

[38] JANG, J., KONG, S., KIM, M., KIM, D., AND KANG, B. B. Secret: Secure channel between rich execution environment and trusted execution environment. In *NDSS* (2015).

[39] KLEIN, G., ANDRONICK, J., ELPHINSTONE, K., MURRAY, T., SEWELL, T., KOLANSKI, R., AND HEISER, G. Comprehensive formal verification of an os microkernel. *ACM Transactions on Computer Systems (TOCS) 32*, 1 (2014), 2.

[40] KLEIN, G., ELPHINSTONE, K., HEISER, G., ANDRONICK, J., COCK, D., DERRIN, P., ELKADUWE, D., ENGELHARDT, K., KOLANSKI, R., NORRISH, M., ET AL. sel4: Formal verification of an os kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles* (2009), ACM, pp. 207–220.

[41] KWON, Y., DUNN, A. M., LEE, M. Z., HOFMANN, O. S., XU, Y., AND WITCHEL, E. Sego: Pervasive trusted metadata for efficiently verified untrusted system services. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (2016), ACM, pp. 277–290.

[42] LI, W., LI, H., CHEN, H., AND XIA, Y. Adattester: Secure online mobile advertisement attestation using trustzone. In *MobiSys* (2015).

[43] LI, W., MA, M., HAN, J., XIA, Y., ZANG, B., CHU, C.-K., AND LI, T. Building trusted path on untrusted device drivers for mobile devices. In *Proceedings of 5th Asia-Pacific Workshop on Systems* (2014), ACM, p. 8.

[44] LI, Y., MCCUNE, J., NEWSOME, J., PERRIG, A., BAKER, B., AND DREWRY, W. Minibox: A two-way sandbox for x86 native code. In *2014 USENIX Annual Technical Conference (USENIX ATC 14)* (2014), pp. 409–420.

[45] LIU, D., AND COX, L. P. Veriui: Attested login for mobile devices. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications* (2014), ACM, p. 7.

[46] MCCUNE, J. M., LI, Y., QU, N., ZHOU, Z., DATTA, A., GLIGOR, V., AND PERRIG, A. Trustvisor: Efficient tcb reduction and attestation. In *Security and Privacy (SP), 2010 IEEE Symposium on* (2010), IEEE, pp. 143–158.

[47] MORGAN, T. P. Arm servers: Cavium is a contender with thunderx. https://www.nextplatform.com/2015/12/09/arm-servers-cavium-is-a-contender-with-thunderx/, 2015.

[48] PEREZ, R., SAILER, R., VAN DOORN, L., ET AL. vtpm: virtualizing the trusted platform module. In *Proc. 15th Conf. on USENIX Security Symposium* (2006), pp. 305–320.

[49] RAJ, H., SAROIU, S., WOLMAN, A., AIGNER, R., COX, J., ENGLAND, P., FENNER, C., KINSHUMANN, K., LOESER, J., MATTOON, D., ET AL. ftpm: A software-only implementation of a tpm chip.

[50] RAJOVIC, N., CARPENTER, P. M., GELADO, I., PUZOVIC, N., RAMIREZ, A., AND VALERO, M. Supercomputing with commodity cpus are mobile socs ready for hpc. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2013), ACM, p. 40.

[51] RAJOVIC, N., RICO, A., PUZOVIC, N., ADENIYI-JONES, C., AND RAMIREZ, A. Tibidabo: Making the case for an arm-based hpc system. *Future Generation Computer Systems 36* (2014), 322–334.

[52] RATH, J. Baidu deploys marvell arm-based cloud server. `http://www.datacenterknowledge.com/archives/2013/02/28/baidu-deploys-marvell-arm-based-server/`, 2013.

[53] ROEMER, R., BUCHANAN, E., SHACHAM, H., AND SAVAGE, S. Return-oriented programming: Systems, languages, and applications. *ACM Transactions on Information and System Security* (2012).

[54] RUBINOV, K., ROSCULETE, L., MITRA, T., AND ROYCHOUDHURY, A. Automated partitioning of android applications for trusted execution environments. In *ICSE* (2016).

[55] SANTOS, N., RAJ, H., SAROIU, S., AND WOLMAN, A. Using arm trustzone to build a trusted language runtime for mobile applications. In *ASPLOS* (2014), ACM, pp. 67–80.

[56] SANTOS, N., RODRIGUES, R., GUMMADI, K. P., AND SAROIU, S. Policy-sealed data: A new abstraction for building trusted cloud services. In *Usenix Security* (2012).

[57] STEINBERG, U., AND KAUER, B. Nova: a microhypervisor-based secure virtualization architecture. In *Proceedings of the 5th European conference on Computer systems* (2010), ACM, pp. 209–222.

[58] STRACKX, R., YOUNAN, Y., PHILIPPAERTS, P., PIESSENS, F., LACHMUND, S., AND WALTER, T. Breaking the memory secrecy assumption. In *Proceedings of the Second European Workshop on System Security* (2009), ACM, pp. 1–8.

[59] SUN, H., SUN, K., WANG, Y., AND JING, J. Trustotp: Transforming smartphones into secure one-time password tokens. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), ACM, pp. 976–988.

[60] SVERDLIK, Y. Paypal deploys arm servers in data centers. `http://www.datacenterknowledge.com/archives/2015/04/29/paypal-deploys-arm-servers-in-data-centers/`, 2015.

[61] US-CERT/NIST. Cve-2014-4322 in qseecom. https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-4322, 2014.

[62] US-CERT/NIST. Cve-2015-4421 in huawei mate7. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4421, 2015.

[63] US-CERT/NIST. Cve-2015-4422 in huawei mate7. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-4422, 2015.

[64] US-CERT/NIST. Cve-2015-6639 in qseecom. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-6639, 2015.

[65] WANG, X., LAZAR, D., ZELDOVICH, N., CHLIPALA, A., AND TATLOCK, Z. Jitk: A trustworthy in-kernel interpreter infrastructure. In *OSDI* (2014), pp. 33–47.

[66] WANG, X., SUN, K., WANG, Y., AND JING, J. Deepdroid: Dynamically enforcing enterprise policy on android devices. In *Proc. of 18th Annual Network and Distributed System Security Symposium (NDSS)* (2015).

[67] YANG, J., AND SHIN, K. G. Using hypervisor to provide data secrecy for user applications on a per-page basis. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments* (2008), ACM, pp. 71–80.

[68] ZHANG, F., CHEN, J., CHEN, H., AND ZANG, B. Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (2011), ACM, pp. 203–216.

Notes

[1] Though seL4 is not originally designed for TrustZone, we found that many TEE-kernels in the market are based on a port of seL4 due to its provable security.

[2] E.g., by "*MSR TTBR0_EL2, Xt*" in 64-bits ARM.

[3] E.g., by configuring HSCTLR or HCR registers.

[4] Link register of monitor mode, which contains the address of trapping instruction