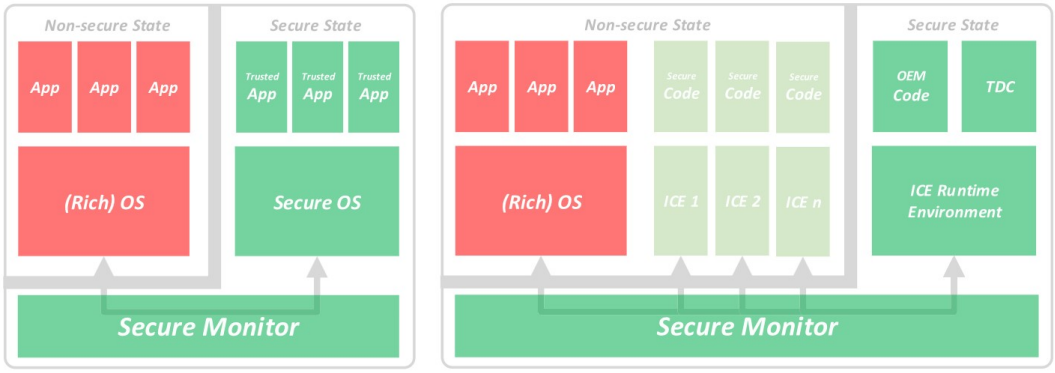


(a) TrustZone architecture (b) TrustICE Architecture

Fig. 2. Architecture comparison between traditional TrustZone's software stack and TrustICE.

from interruptions issued by malicious devices. In summary, the research community has made an effort in allowing generic code to be deployed on the secure domain of TrustZone-enabled processors. Some of these systems aim at reducing the TCB considerably, whilst others support additional features such as secure I/O. Last, some trusted kernels [78, 80] have been designed to complement the security properties of TEE with real-time capabilities, primarily to address the specific requirements of industrial IoT applications.

3.3 Trusted Services for TrustZone-assisted TEE

An alternative approach for exploring the potential of TEE consists not so much in the design of general-purpose TEE kernels, but in building special-purpose *trusted services*. Because such services preclude the need for an underlying OS, they can be engineered in such a way as to offer end-users some specific functionality while depending on a relatively small TCB. Next, we provide an overview of the most relevant trusted services proposed in the literature.

Trusted storage. An important class of trusted service solutions aims to provide secure storage and access to sensitive files in the presence of a potentially compromised local OS. DroidVault [56], for example, introduces the notion of *data vault*, which is an isolated data protection manager running in the trusted domain for secure file management in Android. To achieve this, DroidVault adopts the memory manager and interrupt handler from SierraTEE [102] and is implemented with a data protection manager, an encryption library and a port of a lightweight SSL/TLS library called mbed TLS (formerly known as PolarSSL) [64]. DroidVault supports world switching through software interrupts, secure boot and even inter-world communication. With this trusted service a user can download a sensitive file from an authority and securely store it on the device. The sensitive file is encrypted and signed by the data protection manager before it is stored in the untrusted Android OS. Along the same vein, researchers have studied alternative trusted storage solutions [36, 37, 40], which are not strictly dependent on the Android OS, but whose principled approach allows for a broader adoption across OS platforms.

Authentication and crypto functions. Another relevant category of trusted services aims to provide secure authentication and cryptographic functions. Android Key Store [2], for example, is a security service shipping on Android phones that allows for cryptographic keys to be stored in a container (keystore). The encryption and decryption of the container are performed by the keystore service, which in turn links with a hardware abstraction layer module called "keymaster." The Android Open Source Project (AOSP) provides a software implementation of this module called "softkeymaster," but device vendors can offer support for hardware-based protected storage

by using TrustZone. Stemming from the academia, TrustOTP [107] is a One-Time-Password (OTP) system secured by TrustZone-enabled hardware. Under this system, the OTP is generated based on time and a counter secured by TrustZone's peripheral management. TrustOTP leverages hardware interrupts to trigger the world-switch. Other solutions have leveraged TrustZone to provide trusted services for device-based authentication [9], two-factor authentication [86], and access control [124].

Rich OS introspection and control. Researchers have also explored new ways for leveraging TrustZone to override some functions of the rich OS. Restricted Spaces, a system proposed by Brasser et al. [12], allows for third-parties (hosts) to regulate how users (guests) use their devices (e.g., manage device resources), while in a specific physical space (e.g., at work). To achieve this, Restricted Spaces is capable of securely refining the permissions enforced by the rich OS using a context-aware approach. This system comprises authentication and communication mechanisms between the secure world components of the guest and host. It also supports remote memory operations, which allow for configuration changes such as uninstalling peripheral drivers. This can be done by pointing their interfaces either to NULL or to dummy drivers that just return error codes. TrustDump [108] is a secure memory acquisition tool that allows retrieving the memory content through micro-USB for forensic purposes. Similar to TrustOTP, this system relies on hardware interrupts to trigger world-switches. Both of these systems support trusted user interface (UI) by implementing secure display and input drivers, as well as display controllers to manage the secure framebuffers. A similar approach has been suggested for integrity protection of the rich OS kernel [7, 20] and also for rootkit detection [117].

Trusted UI. Among the most challenging requirements of building trusted services, we find the need to provide secure I/O channels to the user interface. The difficulty lies in that the UI is supported by device drivers of the rich OS, which are both untrusted and difficult to implement with a small code footprint. To address this problem, instead of implementing the required drivers from scratch, some systems allow the secure world domain to reuse untrusted drivers implemented inside the rich OS. In particular, TrustUI [55] excludes from the secure world the device drivers for input, display and network, and reuses the drivers from the normal world, thus achieving a much smaller TCB than previously described systems. Device drivers are split into two parts: a backend and a frontend. The backend runs in the normal world domain and the frontend in the secure world. Both parts rely on proxy modules that run in both worlds and communicate via shared memory. Whenever secure display is necessary, the frontend asks for a framebuffer from the backend driver and sets up that memory region to be secure only, thus isolating the framebuffer from rich OS manipulation. Some systems mentioned above, namely, TrustOTP [107] and TrustDump [108], address this problem by exposing a functionally limited user interface implemented by tiny drivers running in the secure world.

3.4 TrustZone-assisted TEE Systems for the Cloud

Given the proliferation of Arm processors in the mobile device market, existing TrustZone-assisted TEE have been developed primarily to increase the security of data and applications on mobile platforms. While some of these solutions are designed to operate on a standalone basis (e.g., for secure local key storage), other systems have been conceived to be tightly coupled with a cloud backend. A representative example of such a system is DFCLoud [100]. DFCloud aims to leverage a TrustZone-assisted TEE on users' mobile devices to provide secure access control capability to cloud storage services such as Dropbox or Amazon S3. Essentially, the TEE is responsible for managing the cryptographic keys for decrypting the user files stored in encrypted form on the

cloud. By relying on the TEE, an attacker that manages to compromise the local OS will not be able to recover the keys and the content of the files. Before provisioning the keys into the TEE, DFCloud allows the cloud endpoint to remotely attest the client's software thereby ensuring that keys are properly allocated into the TEE rather than to the untrusted domain controlled by the client's OS.

Beyond relying on client-side TEE, researchers have proposed new applications of TrustZone-assisted TEE on the cloud backend itself. Brenner et al. [14] took the first steps at using TrustZone on the cloud by building a TEE-protected privacy proxy for Zookeeper [42]. Zookeeper is a fault-tolerant coordination service for distributed systems that allows the implementation of coordination tasks such as leader election and locks via a filesystem-like client API. This interface allows clients to manage the so-called *znodes* that are payload files and folders simultaneously. The goal of Brenner et al. [14] was to protect the privacy of all data stored inside Zookeeper. This is done by leveraging Zookeeper Privacy Proxies (ZPP) as a lightweight and transparent encryption layer running inside a TEE enabled by TrustZone available on the cloud servers. They place ZPPs in between the Zookeeper clients and the Zookeeper replicas in such a way that client and server implementations remain unmodified. Zookeeper clients connect to ZPPs like they would connect to Zookeeper server replicas and ZPPs connect to real Zookeeper server replicas as traditional clients would. For each client session, a ZPP receives a packet from the client protected using SSL encryption, it then extracts and gathers all the sensitive information, which is then encrypted for secure storage and sent to the real Zookeeper replica.

A second relevant application domain of TrustZone-assisted TEE for cloud has been recently proposed by Brito et al. [15], which aims at enabling secure image processing. Considering cloud services such as Facebook or Instagram, users tend to upload sensitive personal images, which can result in serious privacy violations if leaked from the cloud. While encrypting sensitive content at the client could prevent breaches, oftentimes images need to be decrypted on the cloud servers to be processed, for example, for compression or thumbnail generation. At this point, they can become vulnerable to an adversary with administration privileges. Brito et al. [15] introduced a system named Darkroom, which allows transformation functions to be applied to encrypted user-owned images in a privacy-preserving manner. This goal is achieved by performing such operations inside a TrustZone-assisted TEE at the server-side. Images are decrypted, transformed, and re-encrypted, thereby ensuring that the server's OS never has access to user-image raw data. Brenner et al. [13] have further built on this idea by proposing TrApps, a platform for partitioned applications, targeting an untrusted cloud environment. Similar to Darkroom, the goal of this system is also to reduce the server-side TCB by precluding the need to trust the local OS. TrApps goes beyond Darkroom in the sense that it can support guest general-purpose distributed applications rather than simpler image transformation functions.

A possible barrier for the deployment of TrustZone-assisted TEE in the cloud is the modest popularity of Arm servers among cloud providers. Currently, the data center market is dominated by x86 Xeon and Opteron components manufactured by Intel and AMD, respectively. Nevertheless, some have argued that Arm can become a viable alternative to x86 for servers due to the reduced size, energy efficiency, flexibility, and low cost of Arm processors. Furthermore, the current trends in the evolution of data center workloads seem to suggest that servers will be expected to handle an increasing number of small tasks. When it comes to efficiently accommodating such workload demands, Arm servers emerge as a serious and competitive alternative to existing Intel and AMD servers [41]. By launching the 48-core Centriq 2400 server chip, Qualcomm manifests clear intent to bring out an Arm server chip that can compete with Xeon processors, which suggests that such an evolution in the data center infrastructure is highly plausible.

Table 2. Representative TEE Hardware Technologies

Technology	*Ring*	*I*	*A*	*S*	*SCP*	*MP*	*Ac*	*U*	*ISA*
Intel SGX [22]	3	yes	yes	yes	no	yes	no	++	x86_64
Sanctum [23]	3	yes	yes	yes	yes	no	yes	–	RISC-V
AEGIS [105]	0	yes	yes	yes	no	yes	yes	–	n/a
Bastion [19]	−1	yes	no	yes	no	yes	yes	–	UltraSPARC
AMD SEV [47]	−1	yes	no	no	no	yes	no	+	x86_64
x86 SMM [43]	−2	yes	no	no	no	no	no	+	x86
TrustZone [1]	−2	yes	no	no	no	no	no	++	Arm
TPM [115]	−3	no	yes	yes	n/a	n/a	no	++	n/a
Intel ME [89]	−3	yes	no	no	n/a	n/a	no	+	x86_64

3.5 Alternative TEE Hardware Technologies

In addition to TrustZone, other hardware technologies have been devised to provide basic underlying primitives for the creation of TEE stacks. Although it is not the main scope of this article, we provide a brief comparison of TrustZone against some prominent TEE-enabling technologies and refer the interested reader to specific surveys on this topic [68, 72]. Given the considerable number and variety of existing TEE hardware, we have selected a few representatives based on the protection ring at which the TEE software can be instantiated and built upon. Table 2 lists our selected technologies and presents some of their most interesting features. In particular, it indicates whether or not the hardware provides native mechanisms for isolated execution (I), remote attestation (A), data sealing (S), mitigation of software side-channels focusing on memory access (SCP), and memory protection from physical attacks, e.g., memory or bus probing (MP). We point out if the technology is limited to academic research (Ac), provide our insight about its current usage in real-world computer platforms (U), indicating whether it is widely used (++), seldom used (+) or unused (–), and finally report on the technology's target architecture (ISA).

As far as the protection rings classification is concerned, hardware manufacturers may adopt different nomenclatures, e.g., whereas Intel uses a decreasing numbering system from the least privileged to the most privileged rings, Arm adopts an increasing numbering scheme. Thus, to provide uniform classification across TEE-enabling technologies, we borrow our ring protection terminology from Ning et al. [72], which defines the following levels: ring 3 is for user-level applications, ring 0 for kernel code, ring −1 for hypervisor code, ring −2 for special system maintenance and security functions, and ring −3 for coprocessors and off-processor hardware components. Next, we briefly introduce the TEE-enabling technologies listed in Table 2 according to their respective protection rings.

Ring 3 TEE. One class of TEE hardware allows for securing user space (ring 3) programs without the need to trust in privileged OS code running at ring 0 or below. Intel SGX [22] figures among the most popular of such technologies. This hardware is widely available in processors targeting desktop and server platforms. SGX allows for the creation of memory regions named *enclaves*, which are protected from hardware and software access. Most notably, SGX implements hardware-enabled memory encryption. Sanctum [23] stems from a research initiative targeting RISC-V processors. Similar to SGX, Sanctum enables the creation of enclaves at the user level. However, unlike SGX, enclave memory is not encrypted, which makes the system vulnerable to physical attacks on DRAM. However, Sanctum's design improves on SGX's limited ability to defend against side-channel attacks. As of this writing, Sanctum has not yet been adopted for commercial use.

Ring 0 TEE. Another class of TEE hardware aims to implement secure execution environments at the OS level (ring 0). AEGIS [105] was one of the first TEE hardware architectures to be proposed. In AEGIS, part of the OS is split and runs inside a protected environment established by the processor. This OS partition—named Security Kernel—is responsible for the maintenance of Tamper-Evident Environments (TEs) where security-sensitive programs can be executed. TEs can detect memory tampering attempts by malicious applications or by the untrusted OS. Alternatively, AEGIS can be fully implemented in hardware. In this case, there is no need to provide a Security Kernel, since TE protection can be fully enforced by the hardware.

Ring –1 TEE. Whenever a TEE hardware provides mechanisms to instantiate a TEE stack based on a trusted hypervisor, we say that it operates at ring −1. An example is Bastion [19], a security architecture that relies on both a modified processor and a trusted hypervisor to provide confidentiality and integrity protection for security-sensitive software. Bastion includes mechanisms for off-chip memory protection thereby withstanding physical memory attacks. Upon boot, Bastion secures the state of the hypervisor, which henceforth is responsible for protecting arbitrary software modules. More recently, AMD introduced new x86 features for memory encryption. The Secure Encrypted Virtualization (SEV) [47] technology, in particular, is able to encrypt a virtual machine (VM). By relying on a trusted hypervisor, the guest VMs can be used for hosting TEE software stacks.

Ring –2 TEE. Certain TEE hardware technologies implemented by the processor can operate below the hypervisor level in ring −2. Arm TrustZone technology, which we have discussed extensively in this article, can be highlighted as one of its most representative examples. In fact, the virtualization extensions to the Armv8 architecture allow for the deployment of an untrusted hypervisor in the normal world. Enabled by TrustZone, an independent trusted TEE stack can then reside inside the secure world. The x86 System Management Mode (SMM) [43] can be cited as another example of a ring −2 TEE hardware technology. Introduced by Intel in its x86 platforms back in the ’90s, SMM provides a hardware-assisted isolated environment for the execution of system control functions, such as power management. In spite of its numerous limitations, SMM has been adopted for the design of TEEs featuring very small TCB sizes [8].

Ring –3 TEE. Last, we mention a class of TEE hardware that relies on independent coprocessors; hence, we say they allow for the implementation of ring −3 TEE software stacks. The most prevalent of such technologies is the Trusted Platform Module (TPM) [115]. Specified by the Trusted Computing Group (TCG), the TPM consists of a coprocessor, which is typically located on the motherboard. Its primary purpose is to serve for bootstrapping trust on the local platform: it is responsible for storing the software measurements computed during the trusted boot process of the system, and for securely storing cryptographic keys for remote attestation and data sealing operations. In itself, the TPM does not provide the means for executing security-sensitive code in isolation. Instead, it is typically used in tandem with trusted hypervisors or OSes, which will then be responsible for providing confidentiality and integrity protection of such applications. Differently from the TPM, the Intel Management Engine (ME) [89] consists of a micro-computer introduced by Intel in its recent processors. ME can be leveraged as a TEE for hosting security-sensitive code.

3.6 Discussion

As described in the previous sections, TrustZone has been used as a cornerstone hardware technology for enabling TEE on Arm-based platforms. Given the widespread adoption of Arm by the mobile device industry, it is therefore not surprising that most research on TrustZone-enabled

Table 3. TrustZone-assisted TEE Systems Categorization

	Type	*Descr*	*L*	*SUI*	*SS*	*TCB size*	*Supported NSW*	*I-W Comm*
SierraTEE [102]	TK	OP-comp	C	U	yes	unk	Linux, Android, BSD	GP TEE API
OP-TEE [58]	TK	OP-comp	O	U	yes	unk	Linux, VxWorks	GP TEE API
Open-TEE [70]	TK	OP-comp	O	U	unk	unk	Linux, Android	GP TEE API
Genode [53]	TK	small	O	D	yes	unk	Linux	proprietary
Andix OS [31]	TK	small	O	U	yes	unk	Linux	proprietary
Nokia ObC [50]	TK	small	C	D	yes	10kB	Symbian OS	proprietary
TLR [96]	TK	small	C	U	yes*	152.7kLOC	Windows .Net	.Net Rem
T6 [110]	TK	small	O	U	unk	6kLOC	Linux, Android	GP TEE API
TLK [114]	TK	small	C	U	yes	128kB	Android	proprietary
Samsung KNOX [92]	TK	rich	C	D	yes	unk	Android	GP TEE API
TrustICE [109]	TK	unc	C	U	yes	unk	Linux, Android	proprietary
TrustOTP [107]	TS	auth	C	D	yes*	unk	Android	proprietary
Android Key Store [2]	TS	crypto	O	U	yes	unk	Android	proprietary
TrustDump [108]	TS	forens	C	D	no	450LOC	Android	NMI
Brasser et al. [12]	TS	intros	C	U	yes	unk	Android	proprietary
AdAttester [54]	TS	intros	C	D	no	7.4kLOC	Android	proprietary
TrustUI [55]	TS	UI	C	D	no	10kLOC	Android	proprietary
DroidVault [56]	TS	storage	C	U	yes*	unk	Android	proprietary

TEE has targeted the mobile world. For this reason, we dedicate a few more words in discussing the current state of affairs of TEE research in the mobile environment and elaborating on existing open challenges that demand future research.

Table 3 presents a summary of representative mobile TEE systems based on TrustZone characterized according to several dimensions. The field type categorizes existing systems into two main classes: trusted kernels (TK) and trusted services (TS). Trusted kernels provide runtime support for the execution of general-purpose security-sensitive code inside a TEE, and can further be discriminated according to their main design goal: complying with an open standard, featuring a small code footprint, offering rich functionality, or proposing an unconventional TEE architecture. Trusted services (TS), however, implement special-purpose applications inside the secure world and run directly on bare metal. The selected TSs shown in the table implement a range of different applications, such as secure key storage, authentication, forensics, trusted user interfacing, non-secure world introspection, and secure storage. For each presented system, TK or TS, we indicate the current release licence (L), i.e., open-source (O) or closed-source (C), the currently supported normal world OS, the TCB size (whenever available), and some additional noteworthy characteristics of its internal architecture, in particular: the type of inter-world communication interface (I-W Comm), and whether it provides secure user interface (SUI)—undefined (U) or defined (D)—or secure storage (SS). Whenever this information is available, we indicate if the secure storage mechanisms implemented by a given system provides countermeasures to rollback attacks (signalled by an accompanying star symbol).

A common denominator across all these systems is that they all depend on a software component that runs within the secure world. Therefore, it must be designed and implemented with extreme care. In fact, since the secure world code runs with most elevated system privileges, subverting the integrity of such code (e.g., by exploiting a bug) can potentially allow an adversary to elevate its privileges and take over the entire system, including the normal world OS [88, 99]. Hence,

designers of TrustZone-based TEE systems seek to narrow down the system API as much as possible to ensure its correctness; however, this is no easy task. As a general rule, since trusted services are application-specific, they can achieve a higher reduction of the API surface than trusted kernels. For example, TrustDump is triggered simply by a Non-Maskable Interrupt (NMI) interrupt, whereas the TLR exposes a .Net Remote call interface to the normal world via an SMC call. This is why existing standardization bodies such as the GlobalPlatform tend to put so much effort into the design of TEE Client APIs.

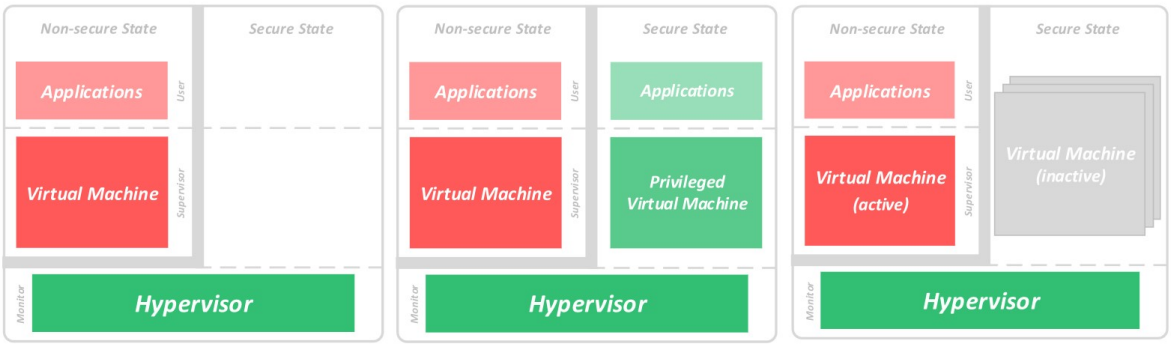
A complementary strategy, perhaps more important than API surface reduction, consists of TCB size reduction. Normally in these systems, the TCB comprises the software components that run with the highest privilege level (e.g., EL3 in Armv8-A) and possibly additional components. Shrinking the TCB size aims to reduce the amount of code that needs to be correctly designed and implemented to ensure the security properties of the system. While this goal is desirable, a negative side-effect emerges, namely, a regression in the functionality offered to users. From this tension between security-utility different solutions have emerged. The advantage of trusted services is a smaller TCB inherent to the sole implementation of strictly necessary components and features. However, trusted kernels require additional runtime support to generic code execution, which in turn means it needs additional code, thus a larger TCB. Keeping a small TCB becomes even more difficult if TKs allow for the execution of non-native code (e.g., Java bytecode or .Net managed code), since the runtime must implement non-native code interpreters while preserving the TCB as small as possible.

In spite of the current advances in the design of secure TEE systems, obtaining strong assurances about the attained security properties remains an open challenge. Such lack of guarantees has fostered some degree of skepticism among device manufacturers who, until the present date, have only deployed TrustZone-based TEE systems in a very conservative manner, oftentimes for delivering specific security functions (e.g., key storage) and/or concealing proprietary software (a.k.a. security through obscurity). Their hesitation is somewhat justified. Just recently, an exploit to a vulnerability in Qualcomm's TrustZone kernel, enabled attackers to bypass Android's full disk encryption mechanism thereby allowing them to retrieve sensitive user data from smartphones [27]. In fact, although the reduction of TCB can help eliminate the presence of potential code vulnerabilities, that, by itself, cannot ensure its correctness. The latest efforts to overcome this challenge have leveraged software verification techniques to formally prove the correctness of privileged code residing within the secure world [29].

4 TRUSTZONE-ASSISTED VIRTUALIZATION

Virtualization technology enables the co-existence of multiple (heterogeneous) environments on the same computing platform. For a long time, virtualization has been used in desktops and servers to optimize resource usage and maximize availability, and, nowadays, this technology starts becoming widespread in mobiles and embedded devices [83, 101]. A virtualized environment consists of three main components: a hardware platform, which provides the hardware resources to deploy the system; a hypervisor, also known as virtual machine monitor (VMM), which virtualizes the hardware; and one or multiple guest OSes or virtual machines (VMs).

TrustZone technology, although implemented for security purposes, enables a specialized, hardware-assisted, form of system virtualization. With a virtual hardware support for dual world execution, as well as other TrustZone features like memory segmentation, it is possible to provide time and spatial isolation between execution environments. Basically, the non-secure software runs inside a VM whose resources are completely managed and controlled by a hypervisor running in the secure world. TrustZone-assisted virtualization is not particularly considered full-virtualization neither paravirtualization, because, although guest OSes can run without



(a) Single-guest (b) Dual-guest (c) Multi-guest

Fig. 3. TrustZone-assisted virtualization for Arm application processors (Cortex-A).

modifications on the non-secure world side, they need to co-operate regarding the memory map and address space they are using. According to the existing state of the art, TrustZone-assisted virtualization solutions [32, 66, 82] support three types of system configurations (see Figure 3): single-guest, dual-guest, and multi-guest. At the time of writing of this article, existing TrustZone-assisted virtualization solutions targets, exclusively, Arm application processors (Cortex-A series). So, the remaining of this section describes TrustZone-assisted virtualization for Armv7-A or Armv8-A architectures.

Next, we describe existing solutions related to the single-guest configuration (Figure 3(a)) [26, 32]. In such a configuration the hypervisor runs in the monitor mode, while the guest OS and its applications run in non-secure supervisor and user mode, respectively. In Section 4.2, we start by describing how the dual-guest configuration is implemented, and then we present and discuss related work [66, 79]. In such a configuration, the hypervisor runs in the monitor mode, and the secure guest OS and its applications run in secure supervisor and user mode, respectively. The VM running in the secure world is considered privileged, because in this world there is no isolation between both supervisor and monitor modes. The normal world hosts the (non-privileged) VM, exactly as in the single-guest configuration. Finally, in Section 4.3, we introduce the multi-guest configuration [69, 82]. In this case, unmodified guest OSes are encapsulated between secure and normal worlds: the active VM runs in the non-secure state, while the context of inactive VMs is preserved in a secured memory area. This setup requires the hypervisor to effectively handle shared hardware resources, mainly processor registers, memory, caches, and MMU.

4.1 Single-guest

The single-guest configuration is the simpler system architecture of a TrustZone-assisted virtualization solution. As illustrated in Figure 3(a), the guest OS executes under the non-secure perimeter, and the hypervisor runs in the monitor mode. The hypervisor has a privileged view of the entire system, while the guest OS has limited access to system resources. The TCB of the system is confined to the code running on the secure world side, which means it just depends on the hypervisor size. Memory, devices, and interrupts assigned to the guest OS are configured as non-secure resources and they are directly managed by the guest OS, while the remaining secure resources are under strict supervision of the hypervisor. The guest OS manages its own MMU and cache lines.

There is just a small number of solutions in the literature that implement such configuration. Frenzel et al. [32] propose the use of TrustZone for implementing the Nizza secure architecture [39]. The secure code comprises a small hypervisor and a set of unprivileged components such as secure device drivers and secure services. The normal world includes the non-secure guest OS and

its applications. The non-secure guest OS uses paravirtualized drivers to send requests to access secure resources but has also drivers to access non-secure devices directly, if devices are configured as non-secure accessible. The system was deployed and evaluated on an NVIDIA Tegra 2. Despite the use of a multicore platform, we believe the implemented solution just supported a single-core configuration. Frenzel et al. concluded that the use of TrustZone's virtualization capabilities resulted in a much lower number of required changes to the Linux as a non-secure guest OS when compared to paravirtualization, while the resulting performance overhead ranges from barely measurable to up to 20% depending on the characteristics of the workloads [32].

Douglas [26] describes a thin hypervisor, which is able to secure a single FreeRTOS instance. The low-footprint hypervisor was conceived and designed for generic Arm processors, but it is discussed how it can be implemented and adapted for a TrustZone-enabled platform: the hypervisor can be isolated in the secure world and the FreeRTOS can run in the non-secure perimeter. The guest OS can manage its own memory and virtual address space, as well as independently handle its own exceptions. The hypervisor is responsible for booting the system as well as for the correct assignment of hardware resources to both worlds.

4.2 Dual-guest

The dual-guest OS system is the most used configuration of existing TrustZone-assisted virtualization solutions, due to the precise match existing among the number of consolidated VMs and the number of virtual states supported by the processor. As depicted in Figure 3(b), each guest OS runs inside its independent world, while the hypervisor runs in monitor mode. This configuration has been typically used for mixed-criticality systems, where the real-time functionalities need to be completely isolated from non-real-time interferences. Typically, a real-time OS (RTOS) runs in the secure world, while a general-purpose OS (GPOS) runs in the normal world. Once the privileged software runs in the secure world, the secure guest OS has a full view of the entire system, which means it is part of the TCB of the system. RTOSes typically have a reduced memory footprint, which makes them attractive candidates for such configurations. The majority of existing solutions typically implements an asymmetric or idle scheduler, which dictates the GPOS is specifically scheduled during the idle slots of the RTOS. Memory, devices, and interrupts are typically partitioned once at boot time. The GIC is usually configured for handling secure interrupts as FIQs, and non-secure interrupts as IRQs. As a result of this design decision, the secure guest OS needs to be slightly changed at kernel-level, while the non-secure guest OS runs without modifications. No cache and MMU management operations need to be performed during a world switch.

Cereia et al. proposed an asymmetric virtualization approach for real-time systems exploiting TrustZone [17, 18]. Their proposed solution supports the execution of an RTOS side by side with a GPOS. The system was evaluated on an emulated platform endowed with an ARM1176JZF-S. According to the conducted evaluation, the authors estimated that for a 1 millisecond hypervisor tick, the expected performance overhead of the GPOS is limited to 0.13%; however, the authors omit if the presented results take into account the penalty of memory accesses. We believe it may not be the case, while comparing the overhead of similar solutions described in this section.

SafeG [94], from the TOPPERS Project,[1] consists of an open-source solution that exploits TrustZone hardware extensions to concurrently execute two different environments: a GPOS and an RTOS. The SafeG monitor, which is the most privileged software component, executes in the monitor mode and is responsible for handling the transitions between the RTOS and the GPOS. At the initialization stage, SafeG configures the resources (memory and devices) assigned to the RTOS as secure resources, and the GPOS-related resources as non-secure. Devices can be shared through

[1] http://www.toppers.jp/en/safeg.html.

a mechanism called re-partition [93, 95]. SafeG configures secure devices to generate FIQ interrupts and non-secure devices to generate IRQ interrupts. The first version of SafeG implemented the idle scheduling principle, but later it was also extended with an integrated scheduler [93]. The system has support for several boards including the NXP i.MX6Q and the Altera Cyclone V SoC. Experiments on a PB1176JZF-S board (equipped with a TrustZone-enabled ARM1176JZF processor) demonstrate a worst-case execution time (WCET) of 1.5μs and 1.7μs for RTOS to the GPOS switching and vice-versa, respectively.

Secure Automotive Software Platform (SASP) [49] implements a lightweight virtualization approach that uses TrustZone technology to provide isolation between a control system and an in-vehicle infotainment (IVI) system. The project is a joint venture between the Korea University and the Hyundai Motor Company. SASP uses the features of TrustZone to simultaneously run an RTOS (e.g., AUTOSAR) for running the control software and a GPOS with the IVI software. Each guest OS runs in each world, according to its criticality. The monitor layer, called V-Monitor, is responsible for managing guests, distributing interrupts, managing shared memory, and mediating device access and communication. SASP has support for both single-core and multicore configurations. The GIC distributes interrupts to each world using FIQ and IRQ according to the classic model. Devices are available only for the secure world, which means the GPOS needs to be slightly paravirtualized. The system was deployed and evaluated on an NVIDIA Tegra 3 in a quad-core configuration: one core is dedicated to the RTOS (AUTOSAR 2.0), while the remaining cores run a symmetric multiprocessing (SMP) version of Linux. Experimental results demonstrate the GPOS has a performance degradation within 1% when performing arithmetic operations and within 5% for system call operations.

LTZVisor [81, 83], from the TZVisor Project,[2] is an open-source lightweight TrustZone-assisted hypervisor mainly targeting the consolidation of mixed-criticality systems. Pinto et al. started by proposing a work in progress in Reference [81] and later presented and described a mature version of the hypervisor [83]. LTZVisor implements the classical dual-guest OS configuration: the secure world hosts the RTOS and the hypervisor, while the normal world is assigned to the GPOS. The two guest OSes share the same CPU, but the asymmetric design principle dictates the GPOS is just scheduled when the RTOS is idle while ensuring the RTOS can preempt the execution of the GPOS. Memory, devices, and interrupts are configured and assigned to respective partitions during system initialization and are not shared between the VMs. LTZVisor has support for Armv7-A architecture, but there are on-going activities for extending support for both Armv8-A and Armv8-M architectures. The hypervisor is very minimalist, presenting a memory footprint of just around 3KB. Experimental results (on a Xilinx ZC702) demonstrate that the RTOS does not have any performance penalty and the virtualized GPOS presents a performance degradation of 2% for a 1 millisecond guest-switching rate. LTZVisor-AMP [79] implements support for a supervised asymmetric multi-processing configuration: one core runs in the secure world and hosts the secure software (LTZVisor and RTOS), while the other core runs in the non-secure world and hosts the non-secure software (GPOS). Experiments demonstrate that the multicore extension solves the problem of starvation, which occurs in single-core platforms (when the RTOS does not yield its control of the CPU [71]) while presenting significant performance advantages when the RTOS has a demanding workload.

Schwarz et al. [97] introduced a disruptive virtualization approach for separation, which is able to switch between a virtualized and non-virtualized execution mode through soft reboots. The work is considerably different from the majority of existing TrustZone-assisted virtualization solutions, because it was designed without taking into consideration a particular CPU architecture

[2] http://www.tzvisor.org/.