ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/330696364

Demystifying Arm TrustZone: A Comprehensive Survey

Article *in* ACM Computing Surveys · January 2019

DOI: 10.1145/3291047

CITATIONS
121

READS
7,223

2 authors:

Sandro Pinto
University of Minho
65 PUBLICATIONS **621** CITATIONS

SEE PROFILE

Nuno Santos
Instituto Superior Técnico, Universidade de Lisboa / INESC-ID
40 PUBLICATIONS **637** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

WallScreen View project

reTHINK – Trustful hyper-linked entities in dynamic networks View project

All content following this page was uploaded by Nuno Santos on 22 February 2019.

The user has requested enhancement of the downloaded file.

Demystifying Arm TrustZone: A Comprehensive Survey

SANDRO PINTO, Centro Algoritmi, Universidade do Minho
NUNO SANTOS, INESC-ID, Instituto Superior Técnico, Universidade de Lisboa

The world is undergoing an unprecedented technological transformation, evolving into a state where ubiquitous Internet-enabled "things" will be able to generate and share large amounts of security- and privacy-sensitive data. To cope with the security threats that are thus foreseeable, system designers can find in Arm TrustZone hardware technology a most valuable resource. TrustZone is a System-on-Chip and CPU system-wide security solution, available on today's Arm application processors and present in the new generation Arm microcontrollers, which are expected to dominate the market of smart "things." Although this technology has remained relatively underground since its inception in 2004, over the past years, numerous initiatives have significantly advanced the state of the art involving Arm TrustZone. Motivated by this revival of interest, this paper presents an in-depth study of TrustZone technology. We provide a comprehensive survey of relevant work from academia and industry, presenting existing systems into two main areas, namely, Trusted Execution Environments and hardware-assisted virtualization. Furthermore, we analyze the most relevant weaknesses of existing systems and propose new research directions within the realm of tiniest devices and the Internet of Things, which we believe to have potential to yield high-impact contributions in the future.

CCS Concepts: • **Computer systems organization → Embedded and cyber-physical systems**; • **Security and privacy → Systems security**; *Security in hardware*; Software and application security;

Additional Key Words and Phrases: TrustZone, security, virtualization, TEE, survey, Arm

ACM Reference format:
Sandro Pinto and Nuno Santos. 2019. Demystifying Arm TrustZone: A Comprehensive Survey. *ACM Comput. Surv.* 51, 6, Article 130 (January 2019), 36 pages.
https://doi.org/10.1145/3291047

1 INTRODUCTION

Arm TrustZone consists of hardware security extensions introduced into Arm application processors (Cortex-A) in 2004 [1, 63]. More recently, TrustZone has been adapted to cover the new generation of Arm microcontrollers (Cortex-M) [65, 113]. TrustZone follows a System-on-Chip (SoC) and CPU system-wide approach to security. This technology is centered around the concept of protection domains named *secure world* and *normal world*. The software executed by the

This work has been partially supported by COMPETE: POCI-01-0145-FEDER-007043, by COMPETE 2020/Portugal 2020/União Europeia within the project Mobile Security Ticketing (No. 11388), which is presented by Link Consulting Tecnologias de Informação SA, and by FCT—Fundação para a Ciência e Tecnologia—within the Project Scope: UID/CEC/00319/2013, UID/CEC/50021/2013, SFRH/BSAB/135236/2017, PTDC/EEI-SCR/1741/2014 (Abyss).
Authors' addresses: S. Pinto, Centro Algoritmi, Universidade do Minho, Campus de Azurém, Guimaraes, 4800-058, Portugal; email: sandro.pinto@dei.uminho.pt; N. Santos, Rua Alves Redol, Lisboa, 1000-029 Lisboa, Portugal; email: nuno.santos@inesc-id.pt.
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2019 Association for Computing Machinery.
0360-0300/2019/01-ART130 $15.00
https://doi.org/10.1145/3291047

processor runs either in the secure or non-secure states. On Cortex-A processors, the privileged software referred by the name of secure monitor implements mechanisms for secure context switching between worlds; on Cortex-M processors, there is no secure monitor software and the bridge by both worlds its handled by a set of mechanisms implemented into the core logic. Both worlds are completely hardware isolated and granted uneven privileges, with non-secure software prevented from directly accessing secure world resources. This strong hardware-enforced separation between worlds opens up new opportunities for securing applications and data. In particular, by constraining the operating system (OS) to operate within the boundaries of the normal world, critical applications can reside inside the secure world without the need to rely on the OS for protection.

For several years now, TrustZone has been widely available on commodity mobile devices. Unfortunately, in spite of all its potential for enhancing security, this technology has remained in a state of relative obscurity for quite some time [118, 119]. Manufacturers of TrustZone-enabled SoCs were somewhat reluctant to disclose technical details, oftentimes requiring developers to sign non-disclosure agreements (NDA) [119] before any architectural-related details could be disclosed to them. Research involving TrustZone has been further slowed down by the limited availability of development platforms in which all of TrustZone's capabilities were unlocked. For instance, certain boards were natively programmed to boot the processor directly into the normal world, thereby preventing system developers from deploying code inside the secure world [119]. As a result, for nearly ten years, TrustZone was mostly used by device manufacturers for monetizing proprietary secure services, the security of which was difficult to examine due to their closed nature.

Yet, over the past few years, we have witnessed a growing interest in TrustZone from both academia and industry. TrustZone has been leveraged in many academic research projects and commercial products alike, providing the security foundations for systems such as Samsung Knox [92], Android's Keystore [2], and OP-TEE [58]. Existing projects span across various application domains, notably mobile [52, 109], industry [31, 78], automotive [49], and aerospace [62, 84], and are released under different licensing policies (open-source and proprietary). A dynamic open source community has matured, contributing to the development of various projects based on TrustZone [31, 58, 102], and standardization bodies like the GlobalPlatform [35] have worked toward the definition of common API specifications to promote interoperability across TrustZone-based solutions. The research community, in particular, has been extremely active in exploring new ways to leverage TrustZone as a key-enabler technology for enforcing Trusted Execution Environments (TEEs) [70, 96, 109] and hardware-assisted virtualization [66, 82, 94]. Researchers have also been very keen on studying the security properties of existing TrustZone-based systems and have managed to uncover important vulnerabilities that demand further research to devise effective solutions [67, 99, 112].

An important factor to such a rising interest in TrustZone has been a shift of attitude by major hardware manufacturers, namely Xilinx, which has fostered R&D through the public disclosure of TrustZone technical details and the release of full-featured development boards [120]. Most importantly, this renewed interest can be explained by the potential impact of TrustZone given the widespread adoption of mobile devices and a near-future dissemination of ubiquitous Internet-enabled low-end devices—the so-called Internet of Things (IoT) [6]. Arm processors share the majority of mobile and embedded markets, powering over 60% of all embedded devices and 4.5 billion mobile phones. Arm has further extended TrustZone-support for the tiniest low-end devices, which Arm estimates to reach nearly 1 trillion by 2035 [104]. It is expected that such a plethora of interconnected devices will generate and exchange substantial amounts of data with security-critical and privacy-sensitive content, which tend to attract cybercrime [59, 91]. Similar to competing trusted hardware technologies such as Intel SGX [22], TrustZone can provide

fundamental primitives for securing sensitive data while benefiting from its wide deployment across a large number of mobile and low-end devices.

In this article, we provide a comprehensive study of TrustZone technology. We are driven, on the one hand, by a generalized interest in this technology that, for the reasons expressed above, is foreseeable to prevail for the upcoming years. On the other hand, we are motivated by the absence of systematization of knowledge surrounding this technology. Although there are some works in the literature that partially describe the TrustZone architecture and some of its applications [71, 90], to the best of our knowledge a complete state-of-the-art of TrustZone technology is absent at the time of this writing. We aim at closing this gap, not only by presenting a detailed picture of the most relevant work based on TrustZone but also by providing an insightful discussion on the current limitations and open issues in the use of TrustZone. Furthermore, based on our study and past research experience on TrustZone, we present our vision regarding what we believe to be promising future research directions. In summary, the key contributions of our work are as follows. First, we provide a systematic description of the TrustZone technology itself, covering the main processors and reviewing a few relevant development boards that are fully compatible with and amenable for TrustZone-based systems development. Second, we provide a detailed study of the existing literature, which we structure into two main bodies of work: TEE and hardware-assisted virtualization. Third, we analyze the most relevant reported weaknesses of existing systems and show that additional research is required before TrustZone-based systems can reach full maturity. Fourth, we propose new research directions that seem to us very exciting and potentially resulting in high impact contributions, particularly within the realm of the tiniest IoT devices.

The remainder of this article is organized as follows. Section 2 presents an architectural description of TrustZone technology for both application processors and the new generation of microcontrollers; additionally, some TrustZone-enabled (development) boards are also presented. Sections 3 and 4 provide an in-depth analysis of the state of the art in TrustZone-enabled systems for TEE and virtualization, respectively. Then, Section 5 provides a critical analysis of the most relevant security issues of existing TrustZone systems, and Section 6 discusses promising research directions targeting primarily IoT and cloud environments. Finally, Section 7 concludes this article.

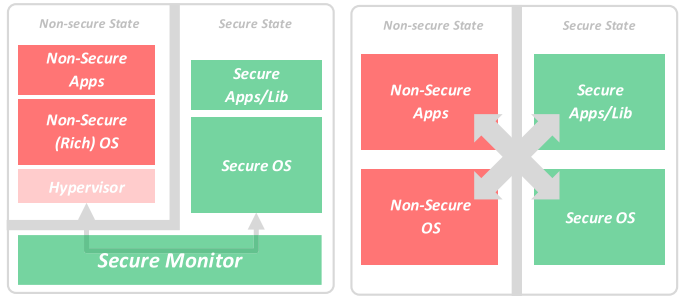
2 TRUSTZONE: HARDWARE AND PLATFORMS

In this section, we provide an overview of TrustZone technology. We start by describing its key architectural features for Arm Cortex-A processors, which can be currently found in a large number of mobile devices (Section 2.1). Then, we highlight the main architectural differences in the new generation Cortex-M processors regarding TrustZone, which was redesigned to accommodate the specific constraints of low-end devices (Section 2.2). Last, we introduce some testbed platforms that can be used for development and research on TrustZone (Section 2.3).

2.1 TrustZone for Application Processors

TrustZone for application processors refers to the hardware-based security built into SoCs to provide a foundation for improved system security for Cortex-A processors [63]. These extensions were added to the Armv6K architecture [1] and introduced significant architectural changes.

The most important architectural change at the processor level consists in the introduction of two protection domains designated by the name of worlds: the secure world and the normal world. Figure 1(a) illustrates these concepts. At a given point in time, the processor operates exclusively in one of these worlds. The world where the processor currently executes is determined by the value of a new 33rd processor bit, also known as the *Non-Secure* (*NS*) bit. The value of this bit can be read from the *Secure Configuration Register* (*SCR*) register, and it is propagated throughout the system down to the memory and peripheral buses. TrustZone introduces an extra processor mode



(a) TrustZone for Cortex-A (b) TrustZone for Cortex-M

Fig. 1. TrustZone technology.

that is responsible for preserving the processor state whenever world transitions occur. This processor mode takes the name of monitor mode, and acts as a bridge for placing the processor in the secure state, independently of the value of the NS bit. A new privileged instruction—*Secure Monitor Call* (*SMC*)—allows for the software stacks residing in both worlds to be bridged by the monitor software. Other than through this instruction, it is possible to enter monitor mode via proper configuration of exceptions, interrupts (IRQ), and fast interrupts (FIQ) handled in the secure world. To reinforce hardware isolation between worlds, the processor has banked versions of the special registers, as well as some system registers (accessed through coprocessor 15 on Armv7-A and using MSR and MRS instructions on Armv8-A). In the normal world, the security-critical system registers and processor core bits are either totally hidden or conditioned by a set of access permissions supervised by the secure world software.

The memory infrastructure has also been extended with TrustZone security features, in particular with the introduction of the TrustZone Address Space Controller (TZASC) and the TrustZone Memory Adapter (TZMA). The TZASC can be used to configure specific memory regions as secure or non-secure, such that applications running in the secure world can access memory regions associated with the normal world, but not the otherwise. Partitioning the DRAM into different memory regions and its respective association with a specific world is performed by the TZASC under the control of a programming interface restricted to the software running with secure world privileges. A similar memory partitioning functionality is implemented by the TZMA, but targeting off-chip ROM or SRAM. Note that the TZASC and the TZMA are optional components defined by the TrustZone specification, which may or may not exist on a specific SoC implementation. Also dependent on the SoC is the granularity at which the memory regions can be specified. Some SoCs with TrustZone extensions include memory controllers that provide limited access to specific memory regions; older SoC implementations do not incorporate such memory controllers at all. Such an example is the Versatile Express platform, which disallows any form of DRAM partitioning into secure and non-secure regions. Modern TrustZone-enabled SoCs, however, come equipped with fully functional TrustZone-enabled memory controllers. Xilinx Zynq and NXP i.MX6 are just a few examples of SoCs that provide full support for TrustZone technology. The TrustZone-aware Memory Management Unit (MMU) allows for each world to have its own virtual-to-physical memory address translation tables. To provide memory isolation at the cache-level, the cache line tags of the processor contain an extra bit that indicates under which world that cache line access has been performed.

TrustZone technology allows for system devices to be restricted to secure or normal worlds. This is achieved with the introduction of a TrustZone Protection Controller (TZPC), which is also an

optional component of the TrustZone specification. The fact that the TZPC is an implementation-specific component leads to diversity in the number and type of TrustZone-aware devices that can be found across hardware platforms. In Xilinx Zynq-based devices, for instance, the Triple Timer Counter 0 (TTC0) cannot be accessed from the normal world, since the TTC0 is permanently restricted to the secure world. The TrustZone architecture extends the Generic Interrupt Controller (GIC) with support for prioritized secure and non-secure sources. Interrupt prioritization is important to prevent denial-of-service (DoS) attacks by non-secure software, since it enables secure interrupts to be handled with higher priority than the non-secure interrupts. Depending on the way the GIC is configured, several interrupt models can be implemented with regard to IRQs and FIQs. Arm proposes to adopt IRQs as interrupt sources pertaining to the normal world and to associate FIQs with interrupt sources from the secure world.

2.2 TrustZone for Microcontrollers

TrustZone technology for Armv8-M [65, 113] has been designed for the new generation of Arm microcontrollers (Cortex-M). At a high level, this variant of TrustZone technology is similar to the variant in Arm Cortex-A processors. In both designs, the processor can execute either in secure or in non-secure state, with non-secure software blocked from accessing secure resources directly. There are, however, important differences between both processor families, namely that Cortex-M has been optimized for faster context switch and low-power applications. In fact, in microcontroller applications, low power consumption, real-time processing, deterministic behavior, and low interrupt latency are mainstream requirements, which lead TrustZone for Armv8-M to be designed from the ground up instead of being reused from Cortex-A processors. As a result, the underlying mechanisms of TrustZone technology for Cortex-M and Cortex-A processors are different.

More specifically, unlike TrustZone technology in Cortex-A processors, the division between worlds in Armv8-M is memory map-based and the transitions take place automatically in exception handling code (see Figure 1). This means that, when running code from the secure memory, the processor state is secure, and, when running code from non-secure memory, the processor state is non-secure. These security states are orthogonal to the existing processor modes, i.e., there are both a Thread and Handler mode in secure and non-secure states. TrustZone technology for Armv8-M excludes the monitor mode and the need for any secure monitor software. This considerably reduces the world switch latency, which translates to more efficient transitions. For bridging software between both worlds, TrustZone now supports multiple secure function entry points, whereas, in TrustZone for Cortex-A processors, the secure monitor handler was the sole entry point. For this purpose, three new instructions were included: secure gateway (SG), branch with exchange to non-secure state (BXNS), and branch with link and exchange to non-secure state (BLXNS). The SG instruction is used for switching from the non-secure to the secure state at the first instruction of a secure entry point; the BXNS instruction is used by secure software to branch or return to the non-secure program; finally, the BLXNS instruction is used by secure software to call non-secure functions. State transitions can also happen due to exceptions and interrupts.

Excepting for stack pointers, in the Armv8-M architecture most of the register file is shared between secure and non-secure states. To separate secure and non-secure stacks, TrustZone-enabled Armv8-M microcontrollers support four physical stack pointers; in this configuration, both security states implements the main stack and the process stack. The starting address of the vector table is determined by a memory-mapped register called the Vector Table Offset Register (VTOR) in the System Control Block (SCB). The VTOR register is banked, which means that one instance exists in each world. Some of the special registers are also banked: the Priority Mask, Control, as well as the Fault Mask Register and the Base Priority registers, just to name a few.

Regarding the memory infrastructure in the Armv8-M architecture, the memory space is also partitioned into secure and non-secure sections. Non-secure addresses are used for memory and peripherals accessible by all software that is running on the device. The secure memory space is further divided into two types: secure and non-secure callable (NSC). Secure addresses are used for memory and peripherals accessible only by secure software. NSC is a special type of secure memory location. This memory area is used to hold SG instructions that allow software to transition between non-secure and secure states. The reason for introducing NSC memory is to prevent other binary data, for example, a lookup table, which has a value the same as the opcode as the SG instruction, from being used as an entry function into the secure state. The security state attributed to each address is determined by the internal Secure Attribution Unit (SAU) or by an external Implementation Defined Attribution Unit (IDAU). The SAU is always present but the number of regions is implementation-specific, while the IDAU is optional and processor-specific. System designers can use an optional IDAU to define a fixed memory map and use an SAU to override the security attributes for some parts of the memory. The SAU can only be programmed in the secure state. The memory partitioning is also used to define peripherals as secure or non-secure. Each world can have a local set of memory access permissions for privileged and unprivileged software. This feature is enabled by the TrustZone-aware Memory Protection Unit (MPU), which provides two distinct MPU interfaces. As in earlier M-series processors, the MPU is an optional component; based on application requirements, designers can exclude the MPU to reduce area and power, or include either a secure or non-secure MPU, or both if necessary. The secure and non-secure MPU can be implemented with a different number of MPU regions.

The Nested Vectored Interrupt Controller (NVIC) was also extended for security. Each interrupt can be configured as secure or non-secure through the Interrupt Target Non-secure register (NVIC_ITNS). This register is only programmable in the secure world. There are no restrictions regarding whether a non-secure or secure interrupt can take place when the processing is running non-secure or secure code. If the arriving exception or interrupt has the same state as the current processor state, then the exception sequence is similar to the previous M-series processors. The main difference occurs when a non-secure interrupt takes place and is handled by the processor during the execution of secure code. In this case, the processor automatically pushes all secure information onto the secure stack and erases the contents from the register banks—this mechanism avoids any leakage of information. Notwithstanding, it is possible to deprioritize non-secure interrupts by setting the PRIS bit field of the Application Interrupt and Reset Control Register (AIRCR) or even avoid handling them while the secure software is running (through the PRIMASK_NS register).

2.3 TrustZone-enabled Hardware Platforms

As TrustZone becomes widespread across all Arm processor families and a key technology for securing small IoT devices, the number of available and cost-efficient TrustZone-enabled (development) platforms seems to follow this trend. Table 1 presents a set of available platforms by comparing them according to five dimensions: name of the platform, designation of the SoC, type of processor, number of cores, and the existence of publicly available TrustZone documentation.

As we can see, a considerable number of the mentioned platforms are endowed with a Xilinx Zynq-7000 SoC. Platforms based on this SoC family have been largely used in both academia and industry, mainly due to its heterogeneity, since it integrates the software programmability of Arm-based processors with the hardware programmability of a Field-Programmable Gate Array (FPGA) [77]. The number of existent Zynq-based development boards is growing: their prices range from less than a hundred (MiniZed) to thousands of US dollars (ZC702). The new generation of Zynq SoCs, the Zynq-based UltraScale+, brings the benefits of the Armv8 architecture and

Table 1. TrustZone-enabled Platforms

Platform	*SoC*	*Processor*	*Multicore*	*Publicly?*
CubieBoard4	Allwinner A80	Cortex-A15/A7	quad-core/quad-core	No
Musca-A1 Board	Arm Musca-A1	Cortex-M33	dual-core	Yes
V2M-Juno r2	Arm Juno (r2)	Cortex-A72/A53	dual-core/quad-core	Yes
SAML11 Xplained Pro	Microchip SAML11	Cortex-M23	single-core	Yes
SAMA5D2-XULT	Microchip SAMA5D2	Cortex-A5	single-core	Yes
MiniZed	Xilinx Zynq-7000	Cortex-A9	single-core	Yes
PYNQ-Z1	Xilinx Zynq-7000	Cortex-A9	dual-core	Yes
ZedBoard	Xilinx Zynq-7000	Cortex-A9	dual-core	Yes
ZYBO	Xilinx Zynq-7000	Cortex-A9	dual-core	Yes
NuMicro M2351	Nuvoton M2351	Cortex-M23	single-core	Yes
Jetson TK1 DevKit	Nvidia Tegra TK1	Cortex-A15	quad-core	No
Jetson TX2 DevKit	Nvidia Jetson TX2	Cortex-A57/Denver	quad-core/dual-core	No
IMX53QSB	NXP i.MX53	Cortex-A8	single-core	Yes
iMX6UL-EVK	NXP i.MX6 UL	Cortex-A7	single-core	Yes
RD-IMX6Q-SABRE	NXP i.MX6	Cortex-A9	quad-core	Yes
MCIMX7-SABRE	NXP i.MX7	Cortex-A7/M4	dual-core/single-core	Yes
Raspberry Pi 3	Broadcom BCM2837	Cortex-A53	quad-core	Yes
R-Car Starter Kit	Renesas R-Car H3	Cortex-A57/A53	quad-core/quad-core	No
ZC702 Eval. Kit	Xilinx Zynq-7000	Cortex-A9	dual-core	Yes
ZCU102 Eval. Kit	Xilinx Zynq UltraScale+	Cortex-A53/R5	quad-core/dual-core	Yes

the power of the 64-bit instruction set; however, at the time of writing of this article, the number of cost-efficient development boards is scarce. So, according to our experience, for those who are interested in building software for mid- to high-end TrustZone-based platforms (Cortex-A), Minized, ZYBO, PYNQ-Z1 or ZedBoard are seen as a good starting point due to the significant number of research papers, open-source projects and technical documentation available, as well as the reasonable selling price. Raspberry Pi 3 is also a very affordable alternative, but the number of available TrustZone-related resources, when compared to the Xilinx Zynq-7000 family, is scarce. NXP also offers interesting platforms, which are widely used by academics and researchers for a reasonable price. Nvidia and Renesas have also presented relevant hardware solutions (e.g., Jetson TX2 DevKit, R-Car Starter Kit Premier), which have been used in industrial settings by some companies [66, 76]. From an academic perspective, these boards are particularly unsuited for TrustZone exploration due to the reluctance of their manufacturers in openly disclosing the technical details regarding their implementation.

Regarding the low-end sector, the market of small IoT devices is still in its infancy. Although Cortex-M23 and Cortex-M33 were announced in Q4 2016, as of this writing, just a few platforms are available on the market. While STMicroelectronics, Renesas, and NXP have not disclosed the technical specifications of TrustZone(-M)-enabled platforms, other manufactureres have taken a different route: Nuvoton has already presented some prototypes of the NuMicro M2351 board, Arm has released the Arm Musca-A1 board, and Microchip has started selling the SAML11 Xplained Pro Evaluation Kit. The NuMicro M2351 features a single-core Cortex-M23 and the SAML11 Xplained Pro Evaluation Kit features a single-core Cortex-M23, while the Musca-A1 test chip features a dual-core Cortex-M33. At the time of the writing of this article, the SAML11 Xplained Pro Evaluation Kit can be purchased for 60 USD; however, the average price for SAML11 Arm Cortex-M23-based

microcontrollers stands around 2 USD, which enables secure small IoT devices (e.g, a smart light bulb or a smart plug) to be implemented at large scale.

3 TRUSTZONE-ASSISTED TEE

In this section, we cover one of the main application areas for TrustZone aimed at the creation of TEE on computer platforms. Next, we start by introducing the TEE concept and explaining how TrustZone is key to realize it. In Sections 3.2 and 3.3, we present, respectively, two design families of TrustZone-based TEE systems: one primarily focused in supporting multiple applications within the TEE, and other on leveraging the TEE to host a single specialized service. Section 3.4 focuses on present efforts do deploy such techniques for cloud clusters and Section 3.5 places TrustZone in perspective against a broader landscape of TEE-enabler hardware technologies. Last, we close this chapter by providing a brief discussion on the main outstanding challenges in the field of TrustZone-assisted TEE.

3.1 TrustZone: A key TEE-enabler Technology

Modern computer systems tend to depend on large trusted computing bases (TCBs). Typically, a TCB comprises complex software such as the OS kernel, privileged services, and libraries. Systems featuring a bloated TCB tend to be more vulnerable to attacks than small-sized TCB systems, because, on the one hand, the likelihood of undetected code vulnerabilities increases as a result of a larger number of lines of source code and more complex inter-component interactions. On the other hand, large TCBs tend to be more exposed to attackers thus opening more doors for effective exploitation of such vulnerabilities. Applications that rely on such complex software platforms inevitably inherit potential security deficiencies of the underlying TCB [111].

To address the TCB bloating problem, TrustZone has appeared as a fundamental hardware mechanism that enables to provide a TEE in which critical applications can execute securely featuring a TCB several orders of magnitude smaller than the rich OS. More specifically, a TEE consists of an isolated environment in which trusted applications can execute without the interference of the local (untrusted) OS. The security properties of a TEE guarantee the confidentiality and integrity of computations that take place inside it. In addition, to enforce isolated execution, a TEE abstraction defines mechanisms for secure provisioning of code and data (including cryptographic keys) into the TEE and trusted channels [116] for retrieving the results of computations and errors. It is also common for a TEE to access some private secure storage space [37] and to allow remote parties to check the integrity and authenticity of the TEE environment through a remote attestation protocol [51]; remote attestation constitutes the basic step for establishing trust between a TEE and a remote party, on top of which secure channels can be established for secure communication to proceed [45, 46].

TrustZone constitutes a natural enabler for building TEE support systems. Essentially, the secure world provides a restricted execution environment where the TEE can reside. Whenever the system boots, the processor enters the secure world to give to any privileged firmware the chance to set up its internal data structures, configure the interrupt controller of the entire system, and set up protections for secure memory regions and peripherals. Upon the completion of these operations, the processor switches worlds and yields control to the bootloader of the rich OS. Since the OS runs in the normal world, it enjoys no privileges to access memory or set up the interrupt table in a way that could gain access to the secure world. A common gateway used to access the secure world is to execute the SMC instruction, which forces the processor to enter into monitor mode. Return to the normal world is performed also through the SMC instruction. From these mechanisms, we can basically run a trusted program inside the secure world without the need to trust the integrity

of the rich OS; if the OS is compromised, attempts to access the secure world address space will result in violations, e.g., resulting in exceptions trapping to the secure monitor.

In addition to world isolation and context-switch capabilities, TrustZone provides building blocks to implement end-to-end security solutions, namely, trusted I/O paths, secure storage, and remote attestation [36, 54, 55]. For trusted I/O paths, TrustZone allows the reflection of the world state of the processor into the peripherals themselves, thereby allowing them to operate in different modes depending on whether the system operates in secure or non-secure states. This is achieved by routing the NS bit state of the processor (which identifies the current world) down to the respective peripheral [120]. In other words, one can configure a device so that data can be routed to/from a specific world. Secure storage can be implemented by installing a data storage component on the device and restrict it to secure world accesses [36]. Likewise, remote attestation is supported by the incorporation of a hardware component (e.g., a Trusted Platform Module) containing trusted code for measuring the integrity of the TEE kernel and unique cryptographic keys [125].

Depending on the trusted program that runs in the secure world, we distinguish two types of TEE architectures: *TEE kernel* or *TEE service*. In the first case, the trusted program implements a basic set of OS functions to manage multiple TEE instances each of them hosting a particular application [96]. The trusted kernel is responsible for: managing memory of the secure word, enforcing memory protection for each TEE, handling communication between TEE and the OS, and providing an API to TEE applications [31, 58, 109]. However, TEE services implement a specific function and do not require any low-level OS logic to manage their own memory and cross-world communication [54, 106]. To prevent mutual interference, only one TEE service can be deployed on the device. This is a disadvantage when compared to TEE kernels, which allow multiple applications to run in independent TEE instances. However, a downside of TEE kernels is that they normally depend on larger TCBs when compared to systems where a single TEE service is deployed. Next, we provide an overview of the state of the art about TEE systems, starting with the solutions based on a TEE kernel architecture, and then focusing on specific- and single-purpose TEE services. Throughout this discussion, we concentrate primarily on solutions targeting mobile platforms. The reason is that the majority of mobile devices are equipped with Arm (application) processors, hence featuring TrustZone technology. In Section 3.4, we cover existing systems and specific challenges when targeting cloud platforms.

3.2 Trusted Kernels for Trustzone-assisted TEE

The basic functionality offered by a TEE system consists of an execution environment where security-sensitive applications can execute in isolation from the rich OS. On TrustZone-enabled platforms, the runtime support for sustaining the lifecycle of such applications is typically provided by a privileged *trusted kernel*, which runs in the secure world. The communication between the rich OS and the trusted kernel requires context switch between worlds. To perform this operation, the rich OS needs to be enhanced with a user-space client API and a TEE device driver responsible for trapping into the trusted kernel.

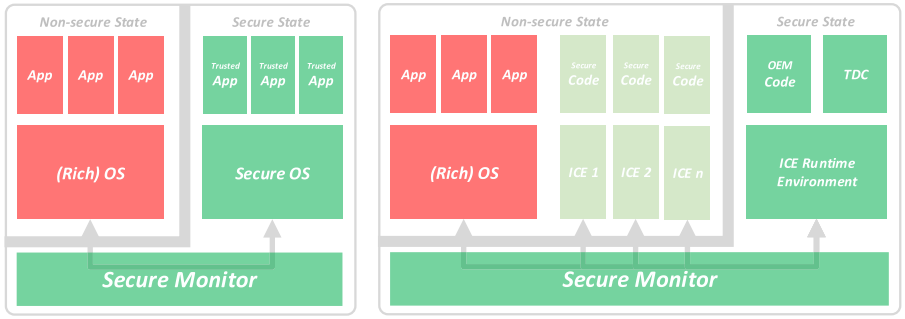
TEE standardization efforts. Given that rich OS and trusted kernels are not necessarily developed by the same manufacturer and nevertheless need to interoperate with each other, a lot of TEE standardization efforts have been advanced. In 2009, the Open Mobile Terminal Platform (OMTP) took some first steps toward this end by specifying a TEE standard that defines a set of security requirements on the functionality a TEE should support [75]. The GlobalPlatform [35] organization went a step further by defining standard APIs: the internal APIs (e.g., TEE Internal API) that a trusted application can rely on and the communication interfaces that rich OS software can use to

interact with the TEE applications maintained by the trusted kernel. The GlobalPlatform has also defined device specifications (e.g., TEE Client API) that TEEs are requested to abide by. Included in these device specifications there is a trusted UI clause (Trusted User Interface API), which states that every GlobalPlatform-compliant device must provide support for a trusted UI. SierraTEE [102], T6 [110], and Open-TEE [70] comply with the GlobalPlatform standard and for this reason allow the development of trusted applications with secure user interfaces. Open-TEE's trusted UI feature is being developed by the community as it was not originally supported.

Rich TEE runtime systems. To gain competitive advantage, some companies build proprietary and closed-source trusted kernels. Samsung KNOX [92] features among the most representative of such systems. KNOX is a defense-grade mobile security platform that aims to provide enterprise data protection with strong guarantees. Security is achieved through several layers of data protection, which include secure boot, TrustZone-based integrity measurement architecture (TIMA) and Security Enhancements for Android (SEAndroid [103]). Samsung KNOX offers a product called KNOX Workspace, which is a container that provides specific mechanisms for isolating and encrypting work data from attackers. This secure container is available on commodity mobile devices and delivers a complete user-friendly environment, which comprises a specific home screen, applications, and widgets. By providing adequate management tools and utilities, this product has been specifically designed to serve the security needs of enterprises.

Small TEE runtime systems. An important drawback of closed systems like KNOX is that it is hard to evaluate whether or not the security properties claimed by its manufacturers are enforced in practice. Furthermore, since KNOX allows for regular rich applications to execute in the secure world, a large number of runtime functions need to be included into KNOX's trusted kernel. As a result, the system's TCB tends to be very large and thus more prone to be exploited than small TEE trusted kernels [5, 25]. To address this problem, the research community has investigated, for some time now, how to build trusted kernels featuring small code footprints. On-board Credentials (ObC) [50, 52] is one of such solutions, originally developed for Nokia mobile devices using the TI M-Shield technology and later ported to TrustZone. ObC supports the development of secure credential and authentication mechanisms. TLR [96] also shares similar goals while in addition providing simple programming abstractions that allow for certain pieces of application code (trustlets) to be instantiated inside the TEE and seamlessly invoked by the application components residing in the normal world. OP-TEE [58], TLK [114], Open-TEE [70], Genode [53], and AndixOS [31] are yet other TEE systems that reduce the TCB of privileged trusted kernel code. With the exception of TLK, these systems have the merit to be publicly available as open source projects that can be used by the research community.

Unconventional trusted kernels. Unlike the aforementioned solutions, such as Samsung KNOX, in which the isolated computing environments reside in the secure world, TrustICE [109] enables the creation of Isolated Computing Environments (ICEs) in the normal world. For this reason, TrustICE's architecture is slightly different from those described above. Figure 2 compares TrustICE's architecture with that of a traditional TrustZone TEE, where trusted applications run inside the secure world. TrustICE works by implementing a trusted domain controller (TDC), which runs in the secure world and is responsible for suspending the execution of the rich OS as well as other ICE's when another ICE is running. Thus, TrustICE supports CPU isolation for running ICEs. For memory isolation, a watermarking mechanism (the TZASC is accessed through Watermark technique on NXP's i.MX53 QSB) prevents the rich OS from accessing code running in the normal world memory belonging to ICE domains. To isolate I/O devices, the secure world blocks all unnecessary external interrupts from reaching the TDC. With the exception of a minimal set of interrupts that allow for trusted UI, this mechanism helps to protect the TDC



(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.

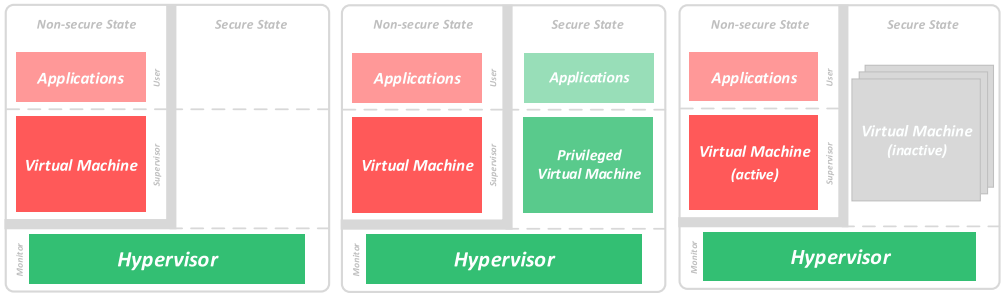
A complementary strategy, perhaps more important than API surface reduction, consists of TCB size reduction. Normally in these systems, the TCB comprises the software components that run with the highest privilege level (e.g., EL3 in Armv8-A) and possibly additional components. Shrinking the TCB size 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/.

or specific hardware extensions. Notwithstanding, the authors discuss how this concept can be implemented with TrustZone technology. The bootloader, the hypervisor, the virtualized (trusted) guest, and secure services can be stored in the secure memory area, and just executes when soft reboot transitions are needed. The execution of the remaining software happens in the normal world. Soft resets would be realized through specific hypercalls (implemented with SMCs).

VOSYSmonitor [66] is a closed-source product developed and maintained by Virtual Open Systems. VOSYSmonitor supports the concurrent execution of two OSes, such as a safety-critical environment and a non safety-critical one. VOSYSmonitor is particularly different from the related work, because it is implemented targeting the Armv8-A architecture and provides the ability to run a hypervisor, such as XEN or KVM [24], on the normal world. The hypervisor running on the normal world is intended to be assisted by Arm VE. VOSYSmonitor natively just has support for dual-guest execution. The scheduler implements the classic idle policy. For a multicore configuration, VOSYSmonitor shares a core between both worlds. VOSYSmonitor configures the secure world (RTOS) and the normal world (GPOS) to handle FIQs and IRQs, respectively. VOSYSmonitor was evaluated on an Arm Juno board and on a Renesas R-Car H3. For a system running FreeRTOS (1ms system tick) and Linux deployed in the Arm Juno board the benchmark reported a performance degradation around 0.5% for a single-core configuration. For the multicore approach, the virtualized system outperforms standalone Linux. No justification was discussed or even pointed out by the authors.

4.3 Multi-guest

For several years the lack of scalability in terms of the number of supported guests was the main reason why several researchers perceived TrustZone as a limited and ill-guided virtualization mechanism [83]. The journey for multi-guest support is a new endeavor. SierraVisor [102], from the OpenVirtualization Project,[3] has introduced support for running multiple OSes concurrently on any TrustZone-enabled Arm11 or Cortex-A9 device; however, SierraVisor lack in providing complete spatial isolation between guests. All non-secure guest OSes share the same non-secure address space, which means that a non-secure guest OS can easily compromise and interfere with the correct execution of the remaining guest OSes. Pinto et al. have been pioneering the real multi-guest support. By changing the security state of memory, devices, and interrupts at runtime, as well as carefully managing shared resources, Pinto et al. [82] demonstrated how several OS instances are able to coexist, completely isolated from each other, on TrustZone-enabled platforms.

As illustrated in Figure 3(c), the hypervisor executes in the monitor mode, while multiple guest OSes are encapsulated between the normal and secure worlds: the active guest OS runs in the normal world, while the context of inactive guests is preserved in the secure world. Since guest OSes are able to run only on the normal world, the system's TCB is limited to the hypervisor size. Memory (including MMU and caches), devices, and interrupts need to be carefully handled by the hypervisor. Memory isolation is ensured by re-configuring the security state of the memory segments during runtime. The active guest OS has its memory space configured as non-secure, while the remaining memory is configured as secure. At every guest switch, the TZASC is re-configured, while MMU and caches need to be flushed. To achieve isolation at the device level, devices assigned to guest partitions are dynamically configured as non-secure or secure, depending on its state (active or inactive), using the TZPC. Interrupts are managed according to a similar strategy: interrupts of secure devices are configured as FIQs, while interrupts of non-secure devices as IRQs. Secure interrupts are redirected to the hypervisor, while non-secure interrupts are directly

[3] http://www.openvirtualization.org/.

sent to the active guest. Interrupts of inactive guest partitions are momentaneously configured as secure, and the processing of such interrupts can follow different approaches [82].

RTZVisor [82] is a monolithic TrustZone-assisted hypervisor that implements strong hardware-enforced isolation between multiple OS instances. The guest OSes are multiplexed on the normal world. Only one active guest OS can run at a time, with inactive OS instances context preserved on the secure area. The strong spatial isolation for memory and devices is ensured through the use of the TZASC and the TZPC, respectively. Temporal isolation is achieved through a cyclic scheduling policy. Non-secure MMU and cache interfaces are shared between the multiple partitions, which requires the hypervisor to perform several maintenance operations each time a new partition is rescheduled. Conducted experiments demonstrate that virtualization overhead is less than 2% for a 10ms guest-switching rate. When the guest switching rate decreases, the overhead increases exponentially, achieving around 8% overhead for a 1ms guest switching rate. This is due to the cache- and MMU-related operations, which need to be performed during the guest switch.

More recently, Martins et al. proposed μRTZVisor [69]. μRTZVisor stands for microkernel real-time TrustZone-assisted hypervisor and is an extended version of RTZVisor [84] for a microkernel-like architecture and an object-oriented implementation. μRTZVisor targets security from the outset, by applying a secure development process. μRTZVisor distinguishes itself from related work, because, as a microkernel-based solution, it is able to run nearly unmodified guest OSes, and, as a TrustZone-assisted solution, it provides a high degree of functionality, configurability, and real-time capabilities. The hypervisor was enhanced with a scheduling policy based on time domains. These time domains can have different priorities and are scheduled according to a preemptive, round-robin schema. The conducted experiments demonstrate, on average, a performance overhead around 2% for a 10ms guest switching rate.

4.4 Discussion

When surveying the literature, TrustZone-assisted virtualization solutions can be divided into three separate groups, depending on the number of VMs they can support. So far, we have described and presented a set of open-source hypervisors that were born inside the academic context, or even closed-source products that some companies have reported in research papers. Notwithstanding, there are still some proprietary solutions available on the market that exploit the hardware extensions of TrustZone technology for virtualization. Such examples include the INTEGRITY Multivisor from Green Hills, the Mentor Embedded Hypervisor from Mentor, and OKL4 Microvisor from Cogs Systems; however, the amount of available information regarding these solutions is scarce, which does not allow us to properly compare and categorize such solutions.

Table 4 summarizes the most important TrustZone-assisted hypervisors by comparing them according to seven dimensions: number of guests (N-G), spatial isolation between guests (SI), multicore support (M-C), real-time guarantees (RT), security measures/guarantees (Se), target processor architecture (PA), and target platform evaluation (PE). The majority of the listed hypervisors has support only for dual-guest OS execution. SierraVisor, RTZVisor, and μRTZVisor introduced support for multi-guest, but SierraVisor uses shadow page tables to provide spatial isolation, requiring modifications at the OS level. The lack of multicore support is also a drawback of the majority of existing solutions. The most active and recent works, such as SafeG, SASP, LTZVisor, and VOSYSmonitor include support for multicore. Notwithstanding, the implemented multicore support follows an asymmetric schema, due to the lower complexity in implementing this approach when comparing to a symmetric multiprocessing schema. The concern in providing a real-time environment is shared among the majority of the solutions. This is strictly related to the fact TrustZone has been seen as an optimal infrastructure for building mixed-criticality systems.

Table 4. TrustZone-assisted Hypervisors Categorization

Hypervisor	*N-G*	*SI*	*M-C*	*RT*	*Se*	*PA*	*PE*
Cereia et al. [17]	dual-guest	Yes	No	Yes	No	Armv6-A	Emulator
Douglas [26]	single-guest	Yes	No	Yes	No	Discussed	Discussed
Frenzel et al. [32]	single-guest	Yes	No	No	Yes	Armv7-A	RealView
LTZVisor [83]	dual-guest	Yes	Yes [79]	Yes	No	Armv7-A, Armv8-(A/M)*	ZC702, ZedBoard
RTZVisor [82]	multi-guest	Yes	No	Yes	No	Armv7-A	ZC702
SafeG [94]	dual-guest	Yes	Yes	Yes	No	Armv6-A, Armv7-A	RealView
SASP [49]	dual-guest	Yes	Yes	Yes	Yes	Armv7-A	Tegra 3
SierraVisor [102]	multi-guest	No	Yes	No	No	Armv7-A	N.A.
VOSYSmonitor [66]	dual-guest	Yes	Yes	Yes	No	Armv8-A	Juno, R-Car H3
μRTZVisor [69]	multi-guest	Yes	No	Yes	Yes	Armv7-A	ZYBO

While for TrustZone-assisted TEEs and trusted services' security has been definitely a major concern, for TrustZone-assisted hypervisors this does not necessarily happen. Apart from the use of a security-oriented technology and hardware security primitives, the majority of existing solutions does not focus on this requirement. Only Frenzel et al. [32], SASP, and recently μRTZVisor, have partially addressed outstanding security issues. Regarding the target processor architecture, Armv7-A is the preferable choice of surveyed hypervisors. This is a consequence of TrustZone-assisted virtualization being seen as the unique hardware-assisted option on those Arm processors were VE are not available.

Despite the evolution of TrustZone-assisted virtualization, existing hypervisors still comprise several limitations, due to the fact that the TrustZone architecture was not designed for virtualization use cases. Most identified drawbacks are related to the memory and device subsystem, as well as to the lack of scalability in terms of the number of guests and cores; the existing open-issues and challenges are strictly linked with the identified limitations.

Spatial isolation. One of the main requirements for virtualization is spatial isolation. TrustZone-assisted virtualization solutions rely on the TZASC and the TZPC for implementing such (memory and devices) isolation. The main drawback of this approach is that, according to TrustZone specification, the existence of the TZASC and the TZPC is not mandatory: both controllers are optional and implementation-specific components of the overall TrustZone architecture. In fact, some TrustZone-enabled SoCs are not endowed with these controllers, and on many others, the TZASC and the TZPC have some constraints, e.g., it is only possible to configure the security state of a subset of memory and devices. Another well-known limitation of the memory subsystem is the absence of a second level memory translation. This limitation places rigid constraints on the memory map, because there is no way to virtualize the physical memory: all guests need to respect the address space other guests are using. Nevertheless, this limitation is seen as an advantage for real-time environments, since the use of virtual memory can hamper with the time predictability of the system. For this reason, Arm decided to introduce support for virtualization in the new Armv8-R architecture adopting Stage-2 MPUs instead of a Stage-2 MMUs [113].

Cache management. For a dual-guest OS configuration, the cache management operations are natively supported by the TrustZone hardware itself, by providing a dual MMU and cache interface. When implementing multi-guest support, non-secure guest OSes need to share the same non-secure cache interface, which means cache-related operations need to be performed at every guest switch. The amount of time needed for cleaning and invalidating L1 and L2 caches can lead

to a significant lack of performance (depending on the size of caches). This means it would be worth using small portions of memory caches and take less time flushing them during the guest switch. An effective and efficient mechanism for managing caches, as well as a deep study for finding a pattern for the optimal cache configuration under a specific set of conditions constitutes an open-research topic for TrustZone-assisted virtualization. Several techniques such as cache-locking [121] and cache-coloring [48] can be investigated further.

Scalability. For several years, TrustZone-assisted virtualization was limited to the coexistence of two VMs, because designers and researchers were not able to realize how to explore the complete TrustZone infrastructure to provide strong spatial isolation between multiple guests. The key for solving this issue was basically to exploit the dynamic features provided by modern TrustZone-enabled controllers. Nevertheless, despite the fact that recently Pinto et al. demonstrated how to tackle and address multiple guest-OSes support, the number of supported VMs continues to be limited from a hardware standpoint. This limitation is not imposed by the amount of available RAM memory but by the granularity of access restrictions on the TZASC. The number of VMs that can be in fact supported in a multi-guest configuration is limited by the number of configurable memory segments supported by the platform.

Another open challenge of TrustZone-assisted virtualization is how to find an effective way to scale multiple guests across multiple cores. The number of existing cores in modern platforms is considerably growing, but on existing TrustZone-assisted hypervisors (i) the multi-guest support is limited to a single-core configuration, while (ii) existing multicore approaches are limited to dual-OS systems. This is because TrustZone provides no means for supporting more than two different states, and, apart from paravirtualizing different guest OSes, there are no means to simultaneously isolate different guests in different cores. This is definitely the main challenge of TrustZone-assisted virtualization, but that should be carefully addressed. Based on our experience, we believe an effective way to simultaneously run multiple isolated guests in multiple cores relying exclusively on TrustZone might be very difficult to implement. We believe efforts for achieving true scalability should go through a synergy between both TrustZone and VE technologies [21].

5 SECURITY ISSUES AND VULNERABILITIES

TrustZone provides several security primitives that developers can leverage to implement trustworthy systems. A simplified but realistic multi-core prototype of the Arm TrustZone technology has been verified and proved to be secure from a hardware standpoint [30]; however, the poor usage of TEEs coupled with some microarchitectural misconceptions have opened several security issues and vulnerabilities. While the former is a consequence of the lack of robust TEE runtime implementations, which results in failing to provide secure containers to applications, the latter is a direct consequence of architectural decisions or the existence of implementation-defined parts on TrustZone specification. Examples of specific microarchitectural attack vectors encompass hardware exceptions (SMC, IRQ, FIQ), caches, and power management modules. The remaining of this section goes through a deeper analysis and description of such issues and vulnerabilities: Section 5.1 focus on TEE-related vulnerabilities and Section 5.2 describes identified hardware-related issues.

5.1 TEE-related Vulnerabilities

As of this writing, according to the National Vulnerability Database (NVD) and several security bulletins (e.g., Qualcomm, Huawei, and Samsung), we found more than 130 vulnerabilities regarding TrustZone and TrustZone-based TEE. Most of these vulnerabilities are related to existing bugs in the TEE kernel and TEE drivers implementation of some providers; a significant number of

registered vulnerabilities are related to the Qualcomm's implementation of the Secure Execution Environment (QSEE). Such vulnerabilities include the lack of input validation, buffer overflows and over-reads, uninitialized variables, and race conditions. At Black Hat 2014, Rosenberg described a vulnerability affecting the QSEE [88]. This vulnerability affected a wide range of TrustZone-enabled mobile devices, including the Samsung Galaxy Note 3, Samsung Galaxy S4, LG Nexus 4 and 5, Moto X, LG G2, and HTC One series. Due to a flaw in bounds-checking SMC requests, an attacker with kernel-level privileges would issue specially crafted SMC requests to cause QSEE to write controlled data to an arbitrary secure memory location. This was exploited to run arbitrary code in the QSEE. The ability to execute arbitrary code in the context of QSEE resulted in the complete compromise of any applications leveraging TrustZone for security guarantees. This vulnerability was exploited to compromise DRM schemes, leak sensitive key materials, defeat OS protection mechanisms, and in some cases (e.g., on some Motorola and HTC devices) manipulate software-programmable fuses to defeat secure boot. At Black Hat 2015, Di Shen [99] described how to exploit the TEE implementation of Huawei devices (HiSilicon SoC), to gain kernel-level privileges in the normal world (privilege escalation) and also execute arbitrary code in the secure world. This vulnerability enabled a non-secure application to get fingerprint images or other encrypted data, to disable signature verification, to load non-trusted modules to the TEE, and even to modify the eFuse data. Also, at Black Hat 2015, Zhang et al. [123] demonstrated how some severe issues existent on several Android fingerprint frameworks could be used to compromise mobile fingerprint systems and get access to protected fingerprints even in a trusted area. Recently, some researchers also unveiled several weaknesses of the Samsung KNOX framework by performing an extensive security analysis [25] and demonstrating several existing vulnerabilities [5].

From a similar perspective, BOOMERANG [67] presented a class of vulnerabilities that arises due to the existence of a semantic gap when passing data between the TEE and the untrusted OS. BOOMERANG is a specific type of attack where a user-level non-secure application can leverage a trusted application to access a portion of memory it does not own. Basically, the malicious application can send specific inputs to a trusted application, which are not properly checked, and then lead the trusted application to manipulate memory locations, which shall not be accessible to the malicious software. Machiry et al. [67] developed a static-analysis tool capable of identifying BOOMERANG vulnerabilities, which helped them to analyze the most popular TEE implementations (QSEE, Kinibi, OP-TEE, SierraTEE, and Huawei) and their trusted applications. They identified BOOMERANG vulnerabilities in four widespread commercial TEE platforms, affecting millions of mobiles. By the time of writing of BOOMERANG's paper, Machiry et al. were already in touch with the TEE vendors to develop specific fixes on their environments. Recently, the Project Zero team at Google have also disclosed a major design issue that affects the security of most devices using QSEE and Kinibi [33].

As aforementioned, the well-known weakness of TrustZone specification in the communication channel is the major venue for exploitation of vulnerabilities of trusted kernels. This happens due to the lack of authentication mechanisms in TrustZone's architecture when the rich execution environment (REE) needs to access secure resources. SeCReT [46] is a framework that implements a secure communication channel used to reinforce the access to trusted resources, by enabling non-secure processes to use session keys. SeCReT provides a session key to a process only when the respective process' code and control flow integrity are verified. To prevent the key from being exposed to attackers, the keys are only readable once, and SeCReT flushes the key as soon as the processor switches into kernel mode. However, authors have recently identified that SeCReT might entail to certain security problems, and proposed TFence framework [45]. TFence removes the kernel dependency when a process communicates with the TEE and provides a direct

communication channel between the client application (non-secure process) and the trusted service [45].

In summary, although the security design of TEEs might be correct, i.e. secure architecture and perfect and robust isolation, the code running inside the TEE may contain vulnerabilities that can be exploited by attackers to corrupt the TEE and compromise the trust state of the entire system. Like Ning et al. [72], we agree that the current state of the art TEE research still lacks frameworks to verify and/or analyze the secure code, properly defense mechanisms within the trusted environments, methods for monitoring and detecting compromised TEEs, and resilient plans to recover and rejuvenate from attacks.

5.2 Hardware-related Vulnerabilities

A number of hardware-related vulnerabilities has also been uncovered over the last few years. The reported vulnerabilities affect different hardware parts of the platform, in particular, the components that constitute the platform's root of trust, caches, power management mechanisms, and FPGAs.

Root of trust. While Intel and AMD specify the TPM as the root of trust for their systems, TrustZone, per se, does not specify where keys for authentication and decryption shall be stored. Several TrustZone-based systems and services proposed in the literature consider the existence of a unique device key, which is used to serve as the root of trust. However, such hardware modules do not always exist on commodity mobile devices, which can result in a lack of guarantees in providing a way to establish trust (authenticity and integrity) in the runtime environment. To address this problem, Zhao et al. [125] proposed the implementation of a root of trust based on Physical Unclonable Functions (PUFs). PUFs are like "digital fingerprints"; they are based on physical variations that occur during semiconductor manufacturing, which can be used to create a unique key (unique identity) using specific fuzzy techniques. Zhao et al. demonstrate the feasibility of their approach by prototyping with a Xilinx Zynq-7000 Evaluation Kit and leveraging the on-chip SRAM, frequently available on mobile devices, to achieve a low-cost and secure root of trust.

Caches. On TrustZone-enabled processors, the cache architecture is modified to include an additional bit that tags the security state of the memory transaction (see Section 2). Even though the secure cache lines are not accessible by the non-secure world, both worlds are equal when competing for the use of cache lines. So, during a world switch, cache lines do not need to be flushed, because a secure cache line fill can evict a non-secure cache line, and vice versa [121]. This cache coherence design improves system performance by eliminating the need to perform cache flushes during world switches; however, it also enables cache contention between the two worlds. Furthermore, to minimize cache pollution (which can be a serious problem for real-time systems [34]), many Arm processors implement a cache-locking feature, which basically prevents cache lines from being evicted. Caches have associated a serious challenge to formal verify programs, because the cache access pattern of security-critical services can lead to secret information leakage. The aforementioned design specificities of TrustZone-enabled caches, although not publicly documented, have been recently observed by several researchers, leveraging recently TrustZone-enabled processors vulnerable to a set of new vector attacks [38, 61, 121, 122].

CacheKit [121] is a rootkit that can bypass memory introspection mechanisms by exploiting existing TrustZone cache incoherences. CacheKit uses the cache-as-RAM technique to guarantee that a malicious portion of code is loaded into the CPU cache, and then it uses cache-locking capability and physical address space to manipulate unused I/O addresses to successfully evade introspection. Despite the fact that Zhang et al. did not validate their approach on various TrustZone-enabled platforms, they discussed the scalability of their solution and demonstrate sincere

confidence regarding the applicability of CacheKit to other platforms. To evaluate their predictions, we performed some experiments regarding the load and lock of specific code in the L2 cache of Zynq-based platforms. We did not replicate the complete attack scenario, but we were able to successfully reproduce both cache-as-RAM and cache-locking techniques.

While CacheKit exploits the existing cache incoherence of TrustZone-enabled devices to evade memory introspection mechanisms, ARMageddon [61], Alias-driven [38], and TruSpy [122] mainly implement a set of cache attacks that allow us to monitor, from the normal world, the cache activity in the secure world and then extract the secret keys stored in the trusted environment. ARMageddon [61] describes a set of cache attacks (e.g., prime and probe, flush and reload) for generic Arm mobile devices. Although not particularly focusing on TrustZone devices, M. Lipp et al. [61] were able to observe that the existing incoherence on TrustZone-enabled caches allows monitoring cache activity in the secure world from the non-secure one. They discussed that, through prime and probe, it is possible to observe cache activity of cryptographic computations within the secure world, which can be used to distinguish whether a provided key is valid or not. Alias-driven [38] describes how cache storage channels can be exploited by means of timing analysis techniques. The proposed cache-based attack vectors exploit self-modifying code and mismatched cacheability attributes ("unexpected cache hit") to subvert confidentiality and integrity properties, allowing an attacker to intentionally place incoherent copies of the same physical address into the caches and consequently measures which addresses are stored or evicted in different levels of cache. Using such attack vector R. Guanciale et al. [38] were able to subvert the integrity properties of a formally verified hypervisor, as well as to extract a private key (128-bit) from an AES encryption service. TruSpy [122] goes a bit further than ARMageddon [61] and Alias-driven [38] and exploits cache contention on TrustZone-enabled processors to implement a timing-based cache side-channel attack. Based on the prime and probe technique, N. Zhang et al. [122] were able to perform two types of attacks: a normal world OS attack (where the attacker has full control of the rich OS) and the normal world Android app attack (where the attacker has zero permissions). Using the T-table-based AES implementation in OpenSSL 1.0.1f as an example, they demonstrate the feasibility of their approach by recovering a full 128 bit AES encryption key. TruSpy is even more powerful than ARMageddon and Alias-driven, because TruSpy does not require kernel privilege and can be performed through a non-privileged Android app.

Power management. Another emerging venue of exploitation goes through a new class of fault attacks that explore the security-obliviousness of energy management mechanisms. Despite the ubiquity of energy management mechanisms on several processors, security is rarely a consideration in the design of such mechanisms due to the complexity of hardware-software needs and the pressure of cost and time-to-market. Tang et al. [112] recently demonstrated that the CLKSCREW attack can be used to break TrustZone-enabled devices by extracting secret cryptographic keys and loading signed applications on commodity mobiles. CLKSCREW attack exploits Dynamic Voltage & Frequency Scaling (DVFS) to push the operating limits of processors until inducing faults. Using only the publicly available information of Nexus 6, they were able to identify the operating limits (frequency and voltage), and then, through software, enable the processor to operate beyond the recommended ones. The CLKSCREW attack requires no further access beyond a malicious kernel driver, thus it can be conducted using just the software control of energy management mechanisms in the target devices. Furthermore, CLKSCREW is more powerful than physical attacks, because it enables fault attacks to be conducted purely from software, opening doors to new remote attacks that do not require physical access to target devices. According to Reference [112], identified vulnerabilities were disclosed to relevant SoC and device vendors, which were very receptive to the disclosure, and promptly started working towards mitigations.

FPGA. As the complexity of current embedded applications grows, the number of heterogeneous SoCs capable of addressing such challenges seems to follow the same trend. Heterogeneous, reconfigurable or sometimes referred to as hybrid platforms combine powerful processing systems (e.g., general-purpose processors and/or microcontrollers, real-time processors, and GPUs) with reconfigurable hardware (e.g., FPGA). This combination enables the efficient configuration of hardware and software components according to the different application needs [77]. Zynq-based SoCs, including the Zynq-7000 and the Zynq UltraScale+, are examples of heterogeneous SoCs that are endowed with TrustZone technology to increase the software security of such SoCs. The problem is that the heterogeneity of such platforms enlarges the attack surface, opening more avenues of exploitation, because a piece of malicious hardware can compromise the secure boot process [44] or even subvert a complete system. Recently, Benhani et al. [11] presented a study about the security evaluation of the TrustZone propagation to FPGA using the Xilinx Zynq-7010 SoC. Benhani et al. found some flaws and weaknesses regarding the security propagation between the processing system (PS) and the programmable logic (PL), which resulted in the successful implementation of six different attacks using small malicious modifications on the programmable logic. Exploiting such flaws they were able to access secure data or even create a DoS attack. These attacks were possible due to the PL was not able to share information regarding the security status of hardware IPs with the PS, and all accesses are approved/denied based on the evaluation of the security status of the AWPROT/ARPROT AXI signals [120].

6 FUTURE DIRECTIONS

Besides addressing the security issues and vulnerabilities discussed in the preceding section, there are several research directions that deserve further exploration. In this section, we elaborate on some possible avenues for investigation on TrustZone focusing primarily on securing and virtualizing the tiniest devices, as well as enabling nested virtualization.

6.1 Securing the Tiniest of Things

The IoT paradigm is making devices smaller, smarter, and increasingly connected [6]. IoT devices are being deployed in massive numbers, and the success of this new wave of the Internet is heavily dependent upon the trust and security built into these billions of different connected devices [59]. Recent attacks on IoT devices have shown that poorly designed connected devices have the ability to bring down key parts of our infrastructures, or even affect our own safety [60]. The problem is that securing IoT devices can be a quandary, with hardware requirements and cost limitations pushing different design directions [74]. To address this problem, Arm decided to span TrustZone to the new generation of microcontrollers, by making security practical at scale and across the entire value chain. With TrustZone built-in on the tiniest of things, Arm is easing the economics of security, reducing risk, cost, and the complexity of implementing robust security measures [3].

As of this writing, the amount of available information regarding the development of secure runtime environments, frameworks, services, or products for Armv8-M is scarce. Currently just a few Armv8-M-based platforms are available on the market (see Section 2.3). Just a few companies such as Prove & Run, Trustonic, and Sequitur Labs are consolidating their position by stepping up in the front of the queue. ProvenCore-M [85], from Prove & Run, is a microkernel implemented using formally proven code. ProvenCore-M for Armv8-M is the next-generation of formally proven ultra-secure TEE, which provides a secure layer running inside the Armv8-M TrustZone-based root of trust. CoreLockr-TZ [98], from Sequitur Labs, is a lightweight service dispatch layer that simplifies accessing security capabilities offered by TrustZone-M. CoreLockr-TZ abstracts complex aspects of the TrustZone-M architecture by presenting a suite of services for easing the access to secure resources and functions by developers writing non-secure applications. Trustonic, a major

player in the TEE industry, has been securing mobile devices with its TEE, Kinibi, since the days of the Samsung S3. Recently, Trustonic has also announced Kinibi-M [87], which is extending its security expertise to small IoT devices. Express Logic has released the X-Ware Secure Platform [28], which implements a set of Express Logic's X-ware components for use with TrustZone-M devices. From a different perspective, CFI CaRE [73] implements a novel control-flow integrity (CFI) mechanism for TrustZone-enabled low-end IoT devices. Finally, ASSURED [4] proposes a secure firmware update framework for the large-scale IoT setting with resource-constrained devices.

Arm is investing strongly on low-end secure devices in terms of specification and standardization, and has recently announced the Platform Security Architecture (PSA) and an accompanying open source software project, named Trusted Firmware-M [3]. The PSA provides a recipe to build a secure system without having to develop all of the elements. While the PSA is architecture agnostic, which means it can be implemented on Cortex-M, Cortex-R, and Cortex-A-based devices, it was mainly designed to secure low-cost IoT devices, where a full TEE would not be appropriate. TrustZone-M provides a reliable and easy method to better implement PSA-defined rules. With PSA, Arm provides an architectural specification, and different partners can provide alternative implementations. We believe this initiative will drive a plethora of projects, products, and research that will bring several new partners to the Arm ecosystem, where not only companies but also academia and hobbyists will play a significant role.

6.2 Virtualizing the Tiniest Devices

While virtualization in embedded systems started by primarily being deployed on high-end devices, the increasing adoption of virtualization technology also starts finding some applicability on low-end hardware, but with several performance limitations due to the lack of hardware support on such devices [16]. To fill this gap, Arm has recently included virtualization extensions in the new generation Cortex-R processors. The Cortex-R52 is the first processor from the Cortex-R family introducing hardware support for virtualization. Hardware virtualization support on real-time processor series is slightly different from the one existing on application processors, due to the need of copying with hard real-time capabilities. OpenSynergy is currently developing a hypervisor assisted by the hardware virtualization support of the Cortex-R52 processor. The hypervisor enables several RTOS and AUTOSAR systems, with different criticality, to run side-by-side on the same platform; however, this domain remains very immature, because hardware platforms endowed with Cortex-R52 processors are still not available on market. To the best of our knowledge, just OpenSynergy is currently developing a hardware-assisted hypervisor for the Cortex-R52.

While the new generation Cortex-R processors only includes hardware virtualization support, the new generation Cortex-M processors introduce TrustZone security extensions. Both processor architectures bring different technologies and target different application domains. Notwithstanding, as TrustZone has enabled an alternative form of system virtualization in Arm application processors, we believe TrustZone can be a game-changer for low-end virtualization. If we agree that TrustZone-assisted virtualization on middle or high-end devices faces difficult challenges of scalability in supporting multiple guests on multicore platforms (see Section 4.4), then we also agree that the pros and cons of each hardware technology, TrustZone or VE, deserve extensive evaluation for use cases requiring a small and fixed number of VMs with real-time requirements. If even for middle- and high-end devices TrustZone might outperform VE for specific use cases (although this is not proved), then on low-end devices it can also happen, especially if we take power consumption as a key-constraint for such devices. Pinto and his research group are currently exploiting TrustZone-M for implementing virtualization in the new generation Cortex-M processors. LTZVisor is being currently ported for the Arm Musca-A1 platform, and the architectural differences of TrustZone for Cortex-M are being studied and evaluated. Their intention

is to support the consolidation of a real-time environment (e.g., FreeRTOS) with an IoT-enabled OSes (e.g., Contiki). This approach will open several opportunities in industrial IoT (IIoT) low-end devices.

6.3 Virtualizing Hypervisors

Traditional single-level virtualization provides the ability to run multiple OSes without modifications inside a VM. The hypervisor, which usually runs on top of the hardware, is responsible for creating a VM environment that is similar to the underlying physical and real hardware [101]. Nested virtualization, in turn, is a technique that provides the means of running a VM inside another VM. Using subsequent levels of virtualization, the hypervisor shall support the execution of multiple other hypervisors with their associated VMs [10]. Nested virtualization is gaining particular attention as new use cases and applications for virtualization are on the rise [57].

Intel has been leading the server and cloud markets with x86 processors for a long time and has introduced hardware support for nested virtualization for several years now. For instance, the IBM Turtles Project [10], introduced in 2010, demonstrated how to run diverse unmodified hypervisors (e.g., VMware, KVM) and OSes (e.g., Linux, Windows) on x86 architectures at a reasonable performance. Arm just recently leveraged its dominance in the embedded and mobile sectors to explore deployments in the cloud infrastructure, and the current need for nested virtualization in such markets lead to the recent introduction of nested virtualization hardware support in the latest Armv8.3-A architecture. Running nested hypervisors on Arm involves running the host hypervisor (i.e., the bare-metal hypervisor that executes directly on top of the hardware) normally at the highest privileged processor mode (i.e., EL2), although modifying the guest hypervisor (i.e., the next level hypervisor) to run in (a deprivileged) EL1, instead of running in EL2. Lim et al. have recently presented a detailed review of Arm nested virtualization while demonstrating that the implementation of nested virtualization on Armv8.3 architectures has associated a significant performance overhead that is considerably worse than in x86 architectures [57].

Despite the current efforts for supporting efficient nested virtualization on Arm architecture are mainly being driven by the addition of NEVE on its next version (Armv8.4-A) [57], we believe TrustZone can also provide an effective foundation for implementing nested virtualization. As TrustZone has been fueling the implementation of several virtualization solutions for Armv7-A processors where VE is not available (see Section 4), a complete synergy between TrustZone and VE can drive the implementation of an alternative form of nested virtualization. The EL3 available in the secure world can be used to run the host hypervisor, while the EL2 available through VE can be used for running the guest hypervisor. Naturally, we are aware of the following: first, that the host hypervisor would need to be modified or designed taking into consideration the non-existence of a Stage-2 or even Stage-3 page tables on the secure world side; and second, that since current Arm processors implement a uniform memory access (UMA) architecture, running more than one host hypervisor at the same time will not be possible, due to absence of memory isolation primitives across multiple EL3 instances. For use-cases requiring just one host hypervisor this strategy seems to be suitable, but for use-cases requiring the coexistence of multiple host hypervisors, it might not be feasible. Nevertheless, this limitation can be overcome if Arm decides to shift for non-uniform memory access (NUMA) architectures, where memory, which is physically isolated, can guarantee the spatial isolation of multiple host hypervisors by itself.

7 CONCLUSION

This article presented a comprehensive survey about TrustZone technology. TrustZone provides strong hardware-enforced isolation for trusted software. The current availability of this technology in today's mobile devices and its expected widespread deployment on tomorrow's tiny smart

devices has raised increasing awareness on TrustZone as a powerful building block for securing end-users' data and applications. This article aims to help researchers and developers getting familiarized with the latest developments around this technology by providing a comprehensive study of the state of the art on TrustZone-based systems for enabling TEE and hardware-assisted virtualization. From our study, we find that there is considerable room for further exploration of this technology, both in terms of devising effective solutions for outstanding security issues and vulnerabilities, and of developing new TrustZone applications primarily for IoT but also the cloud.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their comments. We are mostly grateful to Javier González (CNEX Labs), Jorge Pereira (Prove & Run), Adriano Tavares (Universidade do Minho), and José Martins (Universidade do Minho) for their valuable input and suggestions to improve this article.

REFERENCES

[1] T. Alves and D. Felton. 2004. TrustZone: Integrated hardware and software security. *Tech. In-Depth* 3, 4 (2004), 18–24.
[2] Android. 2018. Android Key Store. Retrieved from https://developer.android.com/training/articles/keystore.html.
[3] Arm Ltd. 2017. Arm Platform Security Architecture Overview. White Paper (Revision 1.1).
[4] N. Asokan, T. Nyman, N. Rattanavipanon, A.-R. Sadeghi, and G. Tsudik. 2018. ASSURED: Architecture for secure software update of realistic embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37, 11 (2018), 2290–2300. DOI: 10.1109/TCAD.2018.2858422.
[5] A. Atamli-Reineh, R. Borgaonkar, R. Balisane, G. Petracca, and A. Martin. 2016. Analysis of trusted execution environment usage in samsung KNOX. In *Proceedings of the Workshop on System Software for Trusted Execution*. ACM, 7:1–7:6. DOI: https://doi.org/10.1145/3007788.3007795
[6] L. Atzori, A. Iera, and G. Morabito. 2010. The internet of things: A survey. *Comput. Netw.* 54, 15 (2010), 2787–2805. DOI: https://doi.org/10.1016/j.comnet.2010.05.010
[7] A. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen. 2014. Hypervision across worlds: Real-time kernel protection from the ARM TrustZone secure world. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 90–102. DOI: https://doi.org/10.1145/2660267.2660350
[8] A. Azab, P. Ning, and X. Zhang. 2011. SICE: A hardware-level strongly isolated computing environment for x86 multi-core platforms. In *Proceedings of the ACM Conference on Computer and Communications Security*. 375–388.
[9] R. Balisane and A. Martin. 2016. Trusted execution environment-based authentication gauge (TEEBAG). In *Proceedings of the New Security Paradigms Workshop*. ACM, 61–67. DOI: https://doi.org/10.1145/3011883.3011892
[10] N. Ben-Yehuda, M. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. 2010. The turtles project: Design and implementation of nested virtualization. In *Proceedings of the USENIX Conference on Operating Systems Design and Implementation*. USENIX Association.
[11] E. M. Benhani, C. Marchand, A. Aubert, and L. Bossuet. 2017. On the security evaluation of the ARM TrustZone extension in a heterogeneous SoC. In *Proceedings of the IEEE International System-on-Chip Conference*. 108–113. DOI: https://doi.org/10.1109/SOCC.2017.8226018
[12] F. Brasser, D. Kim, C. Liebchen, V. Ganapathy, L. Iftode, and A.-R. Sadeghi. 2016. Regulating ARM TrustZone devices in restricted spaces. In *Proceedings of the International Conference on Mobile Systems, Applications, and Services*. ACM, 413–425. DOI: https://doi.org/10.1145/2906388.2906390
[13] S. Brenner, D. Goltzsche, and R. Kapitza. 2017. TrApps: Secure compartments in the evil cloud. In *Proceedings of the International Workshop on Security and Dependability of Multi-Domain Infrastructures*. ACM, Article 5, 6 pages. DOI: https://doi.org/10.1145/3071064.3071069
[14] S. Brenner, C. Wulf, and R. Kapitza. 2014. Running ZooKeeper coordination services in untrusted clouds. In *Proceedings of the USENIX Conference on Hot Topics in System Dependability*. USENIX Association, 2–2.
[15] T. Brito, N. Duarte, and N. Santos. 2016. ARM TrustZone for secure image processing on the cloud. In *Proceedings of the IEEE Symposium on Reliable Distributed Systems Workshops*. 37–42. DOI: https://doi.org/10.1109/SRDSW.2016.17
[16] A. Carvalho, V. Silva, F. Afonso, P. Cardoso, J. Cabral, M. Ekpanyapong, S. Montenegro, and A. Tavares. 2016. Full virtualization on low-end hardware: A case study. In *Proceedings of the Annual Conference of the IEEE Industrial Electronics Society*. 4784–4789. DOI: https://doi.org/10.1109/IECON.2016.7794064
[17] M. Cereia and I. C. Bertolotti. 2008. Asymmetric virtualisation for real-time systems. In *Proceedings of the IEEE International Symposium on Industrial Electronics*. 1680–1685. DOI: https://doi.org/10.1109/ISIE.2008.4677005

[18] M. Cereia and I. C. Bertolotti. 2009. Virtual machines for distributed real-time systems. *Comput. Stand. Interfaces* 31, 1 (Jan. 2009), 30–39. DOI: https://doi.org/10.1016/j.csi.2007.10.010

[19] D. Champagne and R. B. Lee. 2010. Scalable architectural support for trusted software. In *Proceedings of the International Symposium on High-Performance Computer Architecture*. 1–12. DOI: https://doi.org/10.1109/HPCA.2010.5416657

[20] R. Chang, L. Jiang, W. Chen, Y. Xiang, Y. Cheng, and A. Alelaiwi. 2017. MIPE: A practical memory integrity protection method in a trusted execution environment. *Cluster Comput.* (2017), 1–13. DOI: https://doi.org/10.1007/s10586-017-0833-4

[21] G. Cicero, A. Biondi, G. Buttazzo, and A. Patel. 2018. Reconciling security with virtualization: A dual-hypervisor design for ARM TrustZone. In *Proceedings of the IEEE International Conference on Industrial Technology*. 1628–1633. DOI: https://doi.org/10.1109/ICIT.2018.8352425

[22] V. Costan and S. Devadas. 2016. Intel SGX explained. *IACR Cryptology ePrint Archive* 2016 (2016), 86.

[23] V. Costan, I. Lebedev, and S. Devadas. 2016. Sanctum: Minimal hardware extensions for strong software isolation. In *Proceedings of the USENIX Security Symposium*. USENIX Association, 857–874.

[24] C. Dall and J. Nieh. 2014. KVM/ARM: The design and implementation of the linux ARM hypervisor. *SIGPLAN Notes* 49, 4 (2014), 333–348. DOI: https://doi.org/10.1145/2644865.2541946

[25] M. Dorjmyagmar, M. Kim, and H. Kim. 2017. Security analysis of samsung knox. In *Proceedings of the International Conference on Advanced Communication Technology*. 550–553. DOI: https://doi.org/10.23919/ICACT.2017.7890150

[26] H. Douglas. 2010. Thin Hypervisor-Based Security Architectures for Embedded Platforms. Master's thesis, Royal Institute of Technology.

[27] ENISA. 2016. Breaking Android's Full Disk Encryption. Retrieved from https://www.enisa.europa.eu/publications/info-notes/breaking-android2019s-full-disk-encryption.

[28] Express Logic. 2016. X-WARE Secure Platform for ARM Cortex-M processors. Retrieved from https://rtos.com/news/express-logics-x-ware-secure-platform-provides-secure-solution-for-information-and-safety-sensitive-iot-devices/.

[29] A. Ferraiuolo, A. Baumann, C. Hawblitzel, and B. Parno. 2017. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the Symposium on Operating Systems Principles*. ACM, 287–305. DOI: https://doi.org/10.1145/3132747.3132782

[30] A. Ferraiuolo, R. Xu, D. Zhang, A. Myers, and G. Suh. 2017. Verification of a practical hardware security architecture through static information flow analysis. *SIGOPS Operat. Syst. Rev.* 51, 2 (2017), 555–568. DOI: https://doi.org/10.1145/3093315.3037739

[31] A. Fitzek, F. Achleitner, J. Winter, and D. Hein. 2015. The ANDIX research OS - ARM TrustZone meets industrial control systems security. In *Proceedings of the IEEE International Conference on Industrial Informatics*. 88–93. DOI: https://doi.org/10.1109/INDIN.2015.7281715

[32] T. Frenzel, A. Lackorzynski, A. Warg, and H. Härtig. 2010. ARM trustzone as a virtualization technique in embedded systems. In *Proceedings of the 12th Real-Time Linux Workshop*.

[33] Gal Beniamini, Project Zero. 2017. Trust Issues: Exploiting TrustZone TEEs. Retrieved from https://googleprojectzero.blogspot.pt/2017/07/trust-issues-exploiting-trustzone-tees.html.

[34] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. 2018. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptogr. Eng.* 8, 1 (2018), 1–27. DOI: https://doi.org/10.1007/s13389-016-0141-6

[35] Global Platform. 2011. Retrieved from https://www.globalplatform.org/.

[36] J. González and P. Bonnet. 2014. *TEE-based Trusted Storage*. Technical Report. IT University Technical Report Series.

[37] J. González and P. Bonnet. 2014. *Versatile Endpoint Storage Security with Trusted Integrity Modules*. Technical Report. IT University Technical Report Series.

[38] R. Guanciale, H. Nemati, C. Baumann, and M. Dam. 2016. Cache storage channels: Alias-driven attacks and verified countermeasures. In *Proceedings of the IEEE Symposium on Security and Privacy*. 38–55. DOI: https://doi.org/10.1109/SP.2016.11

[39] H. Hartig, M. Hohmuth, N. Feske, C. Helmuth, A. Lackorzynski, F. Mehnert, and M. Peter. 2005. The nizza secure-system architecture. In *Proceedings of the International Conference on Collaborative Computing: Networking, Applications and Worksharing*. 10. DOI: https://doi.org/10.1109/COLCOM.2005.1651218

[40] D. Hein, J. Winter, and A. Fitzek. 2015. Secure block device - secure, flexible, and efficient data storage for ARM TrustZone systems. In *Proceedings of the IEEE Trustcom/BigDataSE/ISPA*. 222–229. DOI: https://doi.org/10.1109/Trustcom.2015.378

[41] Z. Hua, J. Gu, Y. Xia, H. Chen, B. Zang, and H. Guan. 2017. vTZ: Virtualizing ARM TrustZone. In *Proceedings of the USENIX Security Symposium*. USENIX Association, Vancouver, BC, 541–556.

[42] P. Hunt, M. Konar, F. Junqueira, and B. Reed. 2010. ZooKeeper: Wait-free coordination for internet-scale systems. In *Proceedings of the USENIX Annual Technical Conference*, Vol. 8. 9.
[43] Intel. 2009. 64 and IA-32 Architectures Software Developer's Manual.
[44] Nisha Jacob, Johann Heyszl, Andreas Zankl, Carsten Rolfes, and Georg Sigl. 2017. *How to Break Secure Boot on FPGA SoCs Through Malicious Hardware*. Springer International Publishing, Cham, 425–442. DOI: https://doi.org/10.1007/978-3-319-66787-4_21
[45] J. Jang and B. B. Kang. 2018. Retrofitting the partially privileged mode for TEE communication channel protection. *IEEE Trans. Depend. Secure Comput.* (2018), 1–1. DOI: https://doi.org/10.1109/TDSC.2018.2840709
[46] J. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang. 2015. SeCReT: Secure channel between rich execution environment and trusted execution environment. In *Proceedings of the Network and Distributed System Security Symposium*.
[47] D. Kaplan, T. Woller, and J. Powell. 2016. AMD Memory Encryption Tutorial. White Paper. Retrived from https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf
[48] H. Kim and R. Rajkumar. 2017. Predictable shared cache management for multi-core real-time virtualization. *ACM Trans. Embed. Comput. Syst.* 17, 1, Article 22 (2017), 27 pages. DOI: https://doi.org/10.1145/3092946
[49] S. W. Kim, C. Lee, M. Jeon, H. Y. Kwon, H. W. Lee, and C. Yoo. 2013. Secure device access for automotive software. In *Proceedings of the International Conference on Connected Vehicles and Expo*. 177–181. DOI: https://doi.org/10.1109/ICCVE.2013.6799789
[50] K. Kostiainen. 2012. *On-board Credentials: An Open Credential Platform for Mobile Devices*. Doctoral Dissertation, Aalto University.
[51] K. Kostiainen, N. Asokan, and J.-E. Ekberg. 2011. Practical property-based attestation on mobile devices. In *Proceedings of the International Conference on Trust and Trustworthy Computing*. Springer-Verlag, 78–92.
[52] K. Kostiainen, J. Ekberg, N. Asokan, and A. Rantala. 2009. On-board credentials with open provisioning. In *Proceedings of the Symposium on Information, Computer, and Communications Security*. ACM, 104–115. DOI: https://doi.org/10.1145/1533057.1533074
[53] Genode Labs. 2014. Genode—An Exploration of ARM TrustZone Technology. Retrieved from http://genode.org/documentation/articles/trustzone.
[54] W. Li, H. Li, H. Chen, and Y. Xia. 2015. AdAttester: Secure online mobile advertisement attestation using TrustZone. In *Proceedings of the Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 75–88. DOI: https://doi.org/10.1145/2742647.2742676
[55] W. Li, M. Ma, J. Han, Y. Xia, B. Zang, C. Chu, and T. Li. 2014. Building trusted path on untrusted device drivers for mobile devices. In *Proceedings of the Asia-Pacific Workshop on Systems*. ACM, 8:1–8:7. DOI: https://doi.org/10.1145/2637166.2637225
[56] X. Li, H. Hu, G. Bai, Y. Jia, Z. Liang, and P. Saxena. 2014. DroidVault: A trusted data vault for android devices. In *Proceedings of the International Conference on Engineering of Complex Computer Systems*. 29–38. DOI: https://doi.org/10.1109/ICECCS.2014.13
[57] J. Lim, C. Dall, S.-W. Li, J. Nieh, and M. Zyngier. 2017. NEVE: Nested virtualization extensions for ARM. In *Proceedings of the Symposium on Operating Systems Principles*. ACM, 201–217. DOI: https://doi.org/10.1145/3132747.3132754
[58] Linaro. 2014. OP-TEE. https://wiki.linaro.org/WorkingGroups/Security/OP-TEE. Accessed: 2018-02-02.
[59] Z. Ling, K. Liu, Y. Xu, Y. Jin, and X. Fu. 2017. An end-to-end view of iot security and Privacy. In *Proceedings of the IEEE Global Communications Conference*. 1–7. DOI: https://doi.org/10.1109/GLOCOM.2017.8254011
[60] Z. Ling, J. Luo, Y. Xu, C. Gao, K. Wu, and X. Fu. 2017. Security vulnerabilities of internet of things: A case study of the smart plug system. *IEEE Internet Things J.* 4, 6 (Dec 2017), 1899–1909. DOI: https://doi.org/10.1109/JIOT.2017.2707465
[61] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, and S. Mangard. 2016. ARMageddon: Cache attacks on mobile devices. In *Proceedings of the USENIX Security Symposium*. 549–564.
[62] R. Liu and M. Srivastava. 2017. PROTC: PROTeCting drone's peripherals through ARM TrustZone. In *Proceedings of the Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*. ACM, 1–6. DOI: https://doi.org/10.1145/3086439.3086443
[63] Arm Ltd. 2009. ARM Security Technology: Building a Secure System using TrustZone Technology.
[64] Arm Ltd. 2015. mbed TLS. Retrieved from https://tls.mbed.org/.
[65] Arm Ltd. 2017. TrustZone technology for ARMv8-M Architecture. Version 2.0.
[66] P. Lucas, K. Chappuis, M. Paolino, N. Dagieu, and D. Raho. 2017. VOSYSmonitor, a low latency monitor layer for mixed-criticality systems on ARMv8-A. In *Proceedings of the Euromicro Conference on Real-Time Systems (Leibniz International Proceedings in Informatics)*, Vol. 76. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 6:1–6:18. DOI: https://doi.org/10.4230/LIPIcs.ECRTS.2017.6
[67] A. Machiry, E. Gustafson, C. Spensky, C. Salls, N. Stephens, R. Wang, A. Bianchi, Y. R. Choe, C. Kruegel, and G. Vigna. 2017. BOOMERANG: Exploiting the semantic gap in trusted execution environments. In *Proceedings of the Network and Distributed System Security Symposium*.

[68] P. Maene, J. Götzfried, R. de Clercq, T. Müller, F. Freiling, and I. Verbauwhede. 2018. Hardware-based trusted computing architectures for isolation and attestation. *IEEE Trans. Comput.* 67, 3 (Mar. 2018), 361–374. DOI: 10.1109/TC.2017.2647955

[69] J. Martins, J. Alves, J. Cabral, A. Tavares, and S. Pinto. 2017. μRTZVisor: A secure and safe real-time hypervisor. *Electronics* 6, 4 (2017). DOI: https://doi.org/10.3390/electronics6040093

[70] B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan. 2015. Open-TEE - an open virtual trusted execution environment. In *Proceedings of the IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. 400–407. DOI: https://doi.org/10.1109/Trustcom.2015.400

[71] B. Ngabonziza, D. Martin, A. Bailey, H. Cho, and S. Martin. 2016. TrustZone explained: Architectural features and use cases. In *Proceedings of the IEE Conference on Collaboration and Internet Computing*. 445–451. DOI: https://doi.org/10.1109/CIC.2016.065

[72] Z. Ning, F. Zhang, W. Shi, and W. Shi. 2017. Position paper: Challenges towards securing hardware-assisted execution environments. In *Hardware and Architectural Support for Security and Privacy*. ACM, Article 6. DOI: https://doi.org/10.1145/3092627.3092633

[73] T. Nyman, J.-E. Ekberg, L. Davi, and N. Asokan. 2017. *CFI CaRE: Hardware-Supported Call and Return Enforcement for Commercial Microcontrollers*. Springer International Publishing, Cham, 259–284. DOI: https://doi.org/10.1007/978-3-319-66332-6_12

[74] D. Oliveira, T. Gomes, and S. Pinto. 2018. Towards a green and secure architecture for reconfigurable IoT end-devices. In *Proceedings of the ACM/IEEE International Conference on Cyber-Physical Systems*. 335–336. DOI: https://doi.org/10.1109/ICCPS.2018.00041

[75] Open Mobile Terminal Platform (OMTP). 2009. Advanced Trusted Environment: OMTP TR1. Technical Report (v1.1).

[76] M. Paolino, A. Rigo, A. Spyridakis, J. Fanguede, P. Lalov, and D. Raho. 2015. T-KVM: A trusted architecture for KVM ARM v7 and v8 virtual machines. In *Proceedings of the International Conference on Cloud Computing, GRIDs, and Virtualization*. 39–45.

[77] M. Pena, J. Rodriguez-Andina, and M. Manic. 2017. The internet of things: The role of reconfigurable platforms. *IEEE Industr. Electron. Mag.* 11, 3 (Sept. 2017), 6–19. DOI: https://doi.org/10.1109/MIE.2017.2724579

[78] S. Pinto, T. Gomes, J. Pereira, J. Cabral, and A. Tavares. 2017. IIoTEED: An enhanced, trusted execution environment for industrial IoT edge devices. *IEEE Internet Comput.* 21, 1 (Jan. 2017), 40–47. DOI: https://doi.org/10.1109/MIC.2017.17

[79] S. Pinto, A. Oliveira, J. Pereira, J. Cabral, J. Monteiro, and A. Tavares. 2017. Lightweight multicore virtualization architecture exploiting ARM TrustZone. In *Proceedings of the Annual Conference of the IEEE Industrial Electronics Society*. 3562–3567. DOI: https://doi.org/10.1109/IECON.2017.8216603

[80] S. Pinto, D. Oliveira, J. Pereira, J. Cabral, and A. Tavares. 2015. FreeTEE: When real-time and security meet. In *Proceedings of the IEEE Conference on Emerging Technologies Factory Automation*. 1–4. DOI: https://doi.org/10.1109/ETFA.2015.7301571

[81] S. Pinto, D. Oliveira, J. Pereira, N. Cardoso, M. Ekpanyapong, J. Cabral, and A. Tavares. 2014. Towards a lightweight embedded virtualization architecture exploiting ARM TrustZone. In *Proceedings of the IEEE Conference on Emerging Technology and Factory Automation*. 1–4. DOI: https://doi.org/10.1109/ETFA.2014.7005255

[82] S. Pinto, J. Pereira, T. Gomes, M. Ekpanyapong, and A. Tavares. 2017. Towards a TrustZone-assisted hypervisor for real-time embedded systems. *IEEE Comput. Architect. Lett.* 16, 2 (July 2017), 158–161. DOI: https://doi.org/10.1109/LCA.2016.2617308

[83] S. Pinto, J. Pereira, T. Gomes, A. Tavares, and J. Cabral. 2017. LTZVisor: TrustZone is the key. In *Proceedings of the Euromicro Conference on Real-Time Systems (Leibniz International Proceedings in Informatics)*, Vol. 76. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 4:1–4:22. DOI: https://doi.org/10.4230/LIPIcs.ECRTS.2017.4

[84] S. Pinto, A. Tavares, and S. Montenegro. 2016. Space and time partitioning with hardware support for space applications. *Data Systems in Aerospace, European Space Agency, ESA SP 736* (2016).

[85] Prove & Run. 2017. ProvenCore-M. Retrieved from http://www.provenrun.com/products/provencore-m/.

[86] R. Rijswijk-Deij and E. Poll. 2013. Using trusted execution environments in two-factor authentication: Comparing approaches. In *Proceedings of the Open Identity Summit (Lecture notes in informatics)*, Vol. P-223. Gesellschaft for Informatik, 20–31.

[87] Rob Dyke, Trustonic. 2017. Not just droning on! The rise of Kinibi-M. Retrieved from https://www.trustonic.com/news/blog/not-just-droning-rise-kinibi-m/.

[88] D. Rosenberg. 2014. QSEE trustzone kernel integer overflow vulnerability. In *Proceedings of the Black Hat Conference*.

[89] X. Ruan. 2014. *Platform Embedded Security Technology Revealed: Safeguarding the Future of Computing with Intel Embedded Security and Management Engine* (1st ed.). Apress.

[90] M. Sabt, M. Achemlal, and A. Bouabdallah. 2015. Trusted execution environment: What it is, and what it is not. In *Proceedings of the IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. 57–64. DOI: https://doi.org/10.1109/Trustcom.2015.357

[91] A.-R. Sadeghi, C. Wachsmann, and M. Waidner. 2015. Security and privacy challenges in industrial internet of things. In *Proceedings of the ACM/EDAC/IEEE Design Automation Conference*. 1–6.
[92] Samsung. 2013. White Paper: An Overview of Samsung KNOX. Retrieved from http://www.samsung.com/es/business-images/resource/white-paper/2014/02/Samsung_KNOX_whitepaper-0.pdf.
[93] D. Sangorrín. 2012. *Advanced Integration Techniques for Highly Reliable Dual-OS Embedded Systems*. Ph.D. Dissertation.
[94] D. Sangorrín, S. Honda, and H. Takada. 2010. Dual operating system architecture for real-time embedded systems. In *Proceedings of the International Workshop on Operating Systems Platforms for Embedded Real-Time Applications*. 6–15.
[95] D. Sangorrín, S. Honda, and H. Takada. 2012. *Reliable Device Sharing Mechanisms for Dual-OS Embedded Trusted Computing*. 74–91. DOI: https://doi.org/10.1007/978-3-642-30921-2_5
[96] N. Santos, H. Raj, S. Saroiu, and A. Wolman. 2014. Using ARM TrustZone to build a trusted language runtime for mobile applications. *SIGARCH Comput. Archit. News* 42, 1 (Feb. 2014), 67–80. DOI: https://doi.org/10.1145/2654822.2541949
[97] O. Schwarz, C. Gehrmann, and V. Do. 2014. *Affordable Separation on Embedded Platforms*. Springer International Publishing, Cham, 37–54. DOI: https://doi.org/10.1007/978-3-319-08593-7_3
[98] Sequitur Labs. 2017. CoreLockr-TZ. Retrieved from https://www.sequiturlabs.com/corelockrtz/.
[99] D. Shen. 2015. Exploiting TrustZone on android. In *Proceedings of the Black Hat Conference*.
[100] J. Shin, Y. Kim, W. Park, and C. Park. 2012. DFCloud: A TPM-based secure data access control method of cloud storage in mobile devices. In *Proceedings of the IEEE International Conference on Cloud Computing Technology and Science Proceedings*. 551–556. DOI: https://doi.org/10.1109/CloudCom.2012.6427606
[101] J. Shuja, A. Gani, K. Bilal, A. Khan, S. Madani, S. Khan, and A. Zomaya. 2016. A survey of mobile device virtualization: Taxonomy and state of the art. *Comput. Surveys* 49, 1 (Apr. 2016), 1:1–1:36. DOI: https://doi.org/10.1145/2897164
[102] SierraTEE. 2012. Retrieved from http://www.openvirtualization.org/.
[103] S. Smalley and R. Craig. 2013. Security enhanced (SE) android: Bringing flexible MAC to android. In *Proceedings of the Network and Distributed System Security Symposium*, Vol. 310. 20–38.
[104] Philip Sparks. 2017. The route to a trillion devices. White Paper, ARM.
[105] G. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas. 2003. AEGIS: Architecture for tamper-evident and tamper-resistant processing. In *Proceedings of the Annual International Conference on Supercomputing*. ACM, 160–171. DOI: https://doi.org/10.1145/782814.782838
[106] H. Sun, K. Sun, Y. Wang, and J. Jing. 2015. Reliable and trustworthy memory acquisition on smartphones. *IEEE Trans. Info. Forensics Secur.* 10, 12 (Dec. 2015), 2547–2561. DOI: https://doi.org/10.1109/TIFS.2015.2467356
[107] H. Sun, K. Sun, Y. Wang, and J. Jing. 2015. TrustOTP: Transforming smartphones into secure one-time password tokens. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. ACM, 976–988. DOI: https://doi.org/10.1145/2810103.2813692
[108] H. Sun, K. Sun, Y. Wang, J. Jing, and S. Jajodia. 2014. *TrustDump: Reliable Memory Acquisition on Smartphones*. Springer International Publishing, Cham, 202–218. DOI: https://doi.org/10.1007/978-3-319-11203-9_12
[109] H. Sun, K. Sun, Y. Wang, J. Jing, and H. Wang. 2015. TrustICE: Hardware-assisted isolated computing environments on mobile devices. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE Computer Society, 367–378. DOI: https://doi.org/10.1109/DSN.2015.11
[110] T6 TEE. 2014. Retrieved from https://www.trustkernel.com/en/products/tee/t6.html.
[111] A. Tanenbaum, J. Herder, and H. Bos. 2006. Can we make operating systems reliable and secure? *Computer* 39, 5 (May 2006), 44–51. DOI: https://doi.org/10.1109/MC.2006.156
[112] A. Tang, S. Sethumadhavan, and S. Stolfo. 2017. CLKSCREW: Exposing the perils of security-oblivious energy management. In *Proceedings of the USENIX Security Symposium*. USENIX Association, 1057–1074.
[113] J. Taylor. 2016. Security for the next generation of safe real-time systems. In *Proceedings of Embedded World Conference*.
[114] TLK. 2014. Retrieved from http://www.w3.org/2012/webcrypto/webcrypto-next-workshop/papers/webcrypto2014_submission_25.pdf.
[115] Trusted Computing Group. 2011. TPM Main: Part 1 Design Principles, Version 1.2, Revision 116 ed.
[116] A. Vasudevan, E. Owusu, Z. Zhou, J. Newsome, and J. McCune. 2012. *Trustworthy Execution on Mobile Devices: What Security Properties Can My Mobile Platform Give Me?* Springer, Berlin, 159–178. DOI: https://doi.org/10.1007/978-3-642-30921-2_10
[117] J. Williams. 2015. Inspecting data from the safety of your trusted execution environment. In *Proceedings of the Black Hat Conference*.
[118] J. Winter. 2008. Trusted computing building blocks for embedded linux-based ARM trustzone platforms. In *Proceedings of the ACM Workshop on Scalable Trusted Computing*. ACM, 21–30. DOI: https://doi.org/10.1145/1456455.1456460

[119] J. Winter. 2012. Experimenting with ARM TrustZone—Or: How I met a friendly piece of trusted hardware. In *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*. 1161–1166. DOI: https://doi.org/10.1109/TrustCom.2012.157

[120] Xilinx. 2014. Programming ARM TrustZone Architecture on the Xilinx Zynq-7000 All Programmable SoC. *User Guide, UG1019 (v1.0)*.

[121] N. Zhang, H. Sun, K. Sun, W. Lou, and Y. T. Hou. 2016. CacheKit: Evading memory introspection using cache incoherence. In *Proceedings of the IEEE European Symposium on Security and Privacy*. 337–352. DOI: https://doi.org/10.1109/EuroSP.2016.34

[122] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. Hou. 2016. TruSpy: Cache side-channel information leakage from the secure world on ARM devices. *IACR Cryptology ePrint Archive* (2016), 980.

[123] Y. Zhang, Z. Chen, H. Xue, and T. Wei. 2015. Fingerprints on mobile devices: Abusing and leaking. In *Proceedings of the Black Hat Conference*.

[124] B. Zhao, Y. Xiao, Y. Huang, and X. Cui. 2017. A private user data protection mechanism in TrustZone architecture based on identity authentication. *Tsinghua Sci. Technol.* 22, 2 (Apr. 2017), 218–225. DOI: https://doi.org/10.23919/TST.2017.7889643

[125] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng. 2014. Providing root of trust for ARM TrustZone using on-chip SRAM. In *Proceedings of the International Workshop on Trustworthy Embedded Devices*. ACM, 25–36. DOI: https://doi.org/10.1145/2666141.2666145

Received April 2018; revised August 2018; accepted October 2018

View publication stats