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