Building Trusted and Real Time ARM Guests Execution Environments for Mixed Criticality

With T-KVM, a hypervisor architecture that implements an hardware isolated secure and real time environment

Michele Paolino, Kevin Chappuis, Alvise Rigo, Alexander Spyridakis, Jérémy Fanguède,
Petar Lalov and Daniel Raho
Virtual Open Systems
Grenoble, France
Email: {m.paolino, k.chappuis, a.rigo, a.spyridakis, j.fanguede, p.lalov, s.raho} @virtualopensystems.com

***Abstract*—The new ARMv8 architecture is targeting the server, Network Functions Virtualization (NFV), Mobile Edge Computing (MEC) and In-Vehicle Infotainment (IVI) market segments. At the same time, it will empower Internet of Things (IoT), Cyber Physical Systems (CPS), automotive Electronic Control Units (ECU), avionics and mixed criticality devices. In this context, virtualization is a key feature to enable the cloud delivery model, to implement multitenancy, to isolate different execution environments and to improve hardware/software standardization and consolidation. Since guaranteeing a strict isolation of both the data and the code executed in Virtual Machines (VMs) counts today more than ever, the security of the hypervisor and its guests has become dramatically important. This paper extends Trusted Kernel-based Virtual Machine (T-KVM) [1], an architecture for the KVM-on-ARM hypervisor proposed to satisfy the above market trends, in the direction of an efficient and high performance interrupt management. T-KVM integrates software/hardware components to isolate guest Operating Systems (OSes) and enable Trusted Computing along with mixed criticality in ARM virtual machines. It combines four isolation layers: ARM Virtualization and Security Extensions (also known as ARM VE and TrustZone), GlobalPlatform Trusted Execution Environment (TEE) APIs and SELinux Mandatory Access Control (MAC) security policy. In this paper, the T-KVM architecture and its interrupt management features are described in detail, as well as its key implementation challenges and system security considerations. Lastly, a performance evaluation of the proposed solution is presented.**

Keywords–Trusted KVM; ARMv8 Trusted Computing; ARM Virtualization; Mixed Criticality; Real Time.

I. INTRODUCTION

The use of virtualization in ARM platforms is rapidly increasing due to the deployment of SoCs based on this architecture in different environments such as: servers, Cloud and High Performance Computing (HPC), NFV, MEC, IoT, CPS, smart devices, avionics, automotive, etc.

Virtualization enables multiple OSes to run unmodified on the same hardware, thus sharing system's resources such as memory, CPUs, disks and other devices. These resources are frequently target of specific virtualized environment attacks (e.g., CPU cache [2], memory bus [3] and VM's devices [4] [5]). For this reason, the security of the virtualized systems is critical. Historically, isolation has been used to enhance the security of these systems [6], because it reduces the propagation risks in compromised environments [7].

T-KVM [1] is a novel security architecture for virtualized systems, which adds three isolation layers (ARM TrustZone, GlobalPlatform TEE API and SELinux) on top of the standard VMs isolation provided by the hypervisor and provides support for the concurrent execution of a hypervisor and a real time operating system. The main contribution of this paper is the analysis and benchmark of T-KVM in respect to the interrupt management mechanisms during the concurrent execution of two operating systems into the TrustZone execution worlds. In fact, T-KVM targets to provide a fast and efficient interrupt handler, which allows to meet real-time constraints while providing high performance for the guests applications. For this reason, the ARMv8 interrupt management basics and the possible T-KVM implementations have been described and analyzed in this paper, aiming to compare different interrupt management approaches in order to select the best solution for mixed criticality systems.

As mentioned, T-KVM proposes four isolation layers: KVM, ARM TrustZone, GlobalPlatform TEE and SELinux. The former is considered the most popular hypervisor deployed in OpenStack [8], which is a key solution for Cloud, NFV and HPC computing. KVM for ARM is part of the Linux kernel starting from the version 3.9; it is the Linux component that, exploiting the ARM Virtualization Extension, allows to create a fully-featured virtualization environment providing hardware isolation for CPU, memory, interrupts and timers [9]. TrustZone is an hardware security extension for ARM processors and Advanced Microcontroller Bus Architecture (AMBA) devices [10] designed to drastically improve security inside the ARM ecosystem. The extension starts from the assumption that a system, in order to deliver secure services, has to decouple the resources used for general purpose applications from those that handle security assets. To this end, TrustZone creates two hardware isolated partitions in the system: the Secure and the Non Secure World. While the Non Secure World runs a standard OS with optionally a hypervisor, the Secure World contains, handles and protects all the sensitive data (credit card

information, customers list, passwords, etc.) and code (e.g., real time tasks, ciphers functions, etc.) of the system. These two worlds are linked together through the GlobalPlatform TEE API [11] [12], a set of specifications for a secure Remote Procedure Call (RPC) mechanism between the trusted and non-trusted compartment of the system. At the time of writing, these specifications do not support virtualization, preventing the use of Trusted Platform Module (TPM) services inside virtual machines. This work addresses this limitation proposing a design of a set of virtualization extensions to enable the guest operating systems to make use of TPM services provided by the TrustZone Secure World. The latter T-KVM isolation layer is Security-Enhanced Linux (SELinux), a Mandatory Access Control (MAC) solution, which brings type enforcement, role-based access control and Multi-Level Security (MLS) to the Linux kernel [13]. By means of these, SELinux confines processes in security domains, where the interaction with other processes and files is permitted only if there is a specific SELinux policy rule that allows it.

In T-KVM, the above technologies are combined and adapted to work together, providing high security for guest applications and strong isolation for the Secure World OS/Real-Time OS (RTOS), without the need of specific hardware or software. As a matter of fact, the proposed architecture relies on open source (KVM and SELinux) components, public specifications (GlobalPlatform TEE Internal and Client APIs) and available hardware features (ARM TrustZone and VE). For these reasons, T-KVM can be easily ported to currently available ARM platforms, such as Cloud Infrastructure systems based on OpenStack, automotive ECU, avionics platform and other embedded systems.

The T-KVM architecture matches perfectly with server platforms and user devices but also with mixed criticality systems, where different levels of criticality need to interact and co-exist on a common hardware platform (e.g., infotainment and instrument clusters in avionics, or Advanced Driver Assistance Systems - ADAS - and web browsing in automotive, etc.), because it makes possible to run in the Secure World a critical security sensitive application (e.g., encryption algorithms, real time tasks, etc.) totally isolated from the Normal world.

The remaining part of this paper is organized as follows: Section II introduces the main security components of the proposed architecture. Section III contains details about the T-KVM architecture, its implementation and security considerations. Section IV describes the T-KVM interrupt management for real time and mixed criticality systems while Section V presents a performance analysis of the overhead and interrupt latencies of the proposed solution. The related work is presented in Section VI and Section VII concludes the paper.

II. THE SECURITY COMPONENTS

In this section, the isolation layers, which characterize the T-KVM architecture, are described.

A. KVM hypervisor

A hypervisor is a software layer, which is able to create virtual instances of hardware resources such as CPUs, memory, devices, etc. in order to enable the execution of multiple operating systems on the same hardware. Different implementation approaches lead to different hypervisor types: a type 1 hypervisor, is a bare metal hypervisor, which runs directly on the hardware (XEN or VMWare ESX). A type 2 hypervisor is, on the other hand, a hypervisor, which runs inside an operating system (Oracle VirtualBox or VMWare Workstation) at the application layer. Usually, the latter is used in less critical applications [14] because of its dependency from the underlying operating system.

KVM is a hypervisor included in the Linux kernel and available for ARM, x86 and s390 architectures. It is neither a type 2 hypervisor because it does not run as a normal program inside Linux, nor is a typical type 1 hypervisor, because it relies on the Linux kernel infrastructure to run. KVM exploits the CPU Virtualization Extensions to execute guest's instructions directly on the host processor and to provide VMs with an execution environment almost identical to the real hardware. Each guest is run in a different instance of this execution environment, thus isolating the guest operating system. For this reason, this isolation has been used for security purposes [15] [16] [17] [18] in many scientific works. In the ARM architecture, the KVM isolation involves CPU, Memory, Interrupts and timers [9].

B. TrustZone

ARM TrustZone is a set of hardware security extensions for ARM processors and AMBA devices. With TrustZone, the hardware platform is split in two parts, the Secure and the Non Secure Worlds. In order to isolate these two compartments, TrustZone requires: CPU with ARM Security Extensions (SE) along with TrustZone compliant Memory Management Unit (MMU), AMBA system bus, interrupt and cache controllers. Hence, the isolation provided by TrustZone includes CPU, AMBA devices, interrupts, memory and caches.

The Secure World is considered trusted, and is responsible for the boot and the configuration of the entire system. In fact, the CPU has banked registers for each World, and security specific configurations can be performed in Secure World mode only. This compartment contains the root of trust of the system and protects sensitive data. The access to AMBA peripherals such as fingerprint readers, cryptographic engines, etc. can be restricted only to the Secure World, thus protecting security devices.

On the other hand, the Non Secure World is intended to be the user's World. In this untrusted compartment, a standard operating system (i.e., Android or Linux) is run. Security sensitive operations such as the access to a secret or the interaction with a real time task are provided to the user's application running in this compartment by the services run in the Secure World.

These two compartments interact with each other through a specific CPU mode, namely the Monitor Mode. It typically runs a secure context switch routine and is capable of routing interrupts, depending on the configuration, either to the Secure or Non Secure World.

Moreover, the use of the ARM VE Extensions, and consequently of KVM, is possible only in the Non Secure World.

ARM TrustZone is compliant with the GlobalPlatform TEE System Architecture specification [19], which defines the attributes that the hardware must have to properly execute a TEE.

C. GlobalPlatform TEE

GlobalPlatform specification defines the TEE as an execution environment, which provides security features such as isolated execution, integrity of Trusted Applications (i.e., applications run in the TEE) along with confidentiality of their assets [19]. This is done by means of specific hardware capabilities of the system, such as ARM TrustZone. The TEE protects Trusted Applications (TA) and their data from the Rich Execution Environment (REE), the environment where a standard operating system such as Linux or Android is run. Figure 1 depicts the standard architecture of a GlobalPlatform TEE compliant system.

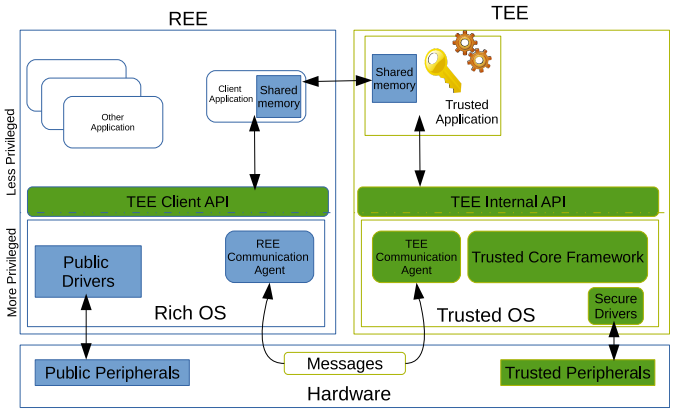

Figure 1. Standard TEE architecture

In order to isolate the TEE from the REE, GlobalPlatform provides a set of specifications, which include the following software components [11] [12]:

- The Trusted Core Framework is a common abstraction layer, which provides to the TEE Internal API OS-like functions, such as memory management, entry points for TAs, panic and cancellation handling, TA properties access, etc.
- The TEE Client API is an Inter Process Communication (IPC) API that deals with the communication between the REE and the TEE. It allows the applications in the REE (Client Applications or CAs) to leverage the services offered by the TEE.
- The (TEE and REE) communication agents provide support for messaging between the CAs and the TEE. They interact with the Monitor mode to request a context switch between the two Worlds.
- The TEE Internal API allows the TAs to leverage the services offered by the TEE through the following APIs: Trusted Storage for Data and Keys, TEE Cryptographic Operations, Time, and TEE Arithmetical.

Lastly, it is worth to mention that the deployment of a GlobalPlatform compliant solution enables the use of existing TA and CA applications. This a is very important factor, especially in an environment such as the embedded trusted computing, where by tradition the security solutions were developed each time from scratch to address new device families.

D. SELinux

SELinux is a software implementation of the MAC security policy available in the Linux kernel as Linux Security Module (LSM). The key feature of MAC is that the access control decisions are not at discretion of individual users, root included [13]. Thus, once the system security policies have been defined and loaded at boot time in the kernel, they can not be modified. In this way, the subject (e.g., a process) access to objects (e.g., file, socket, etc.) is enforced in the system.

The very same concept can be applied to virtual machines using sVirt, which is a feature of the libvirt library. sVirt installs a set of virtualization specific security policies and automatically tags VMs and their resources in order to isolate guest systems. This isolation prevent any access to VM's resources (disk/kernel files, shared memory, etc.) from external subjects (other VMs, the root user, etc.).

For this and for performance reasons [20], the use of SELinux in virtualized systems is encouraged.

III. THE TRUSTED HYPERVISOR: T-KVM

T-KVM is a secure hypervisor architecture based on KVM, which combines a Trusted Computing solution such as TrustZone with GlobalPlatform TEE and SELinux, to enable both Trusted Computing and RT support for ARM based platforms. In Figure 2, all the components described in Section II are shown together, composing the T-KVM architecture.

In T-KVM, the GlobalPlatform TEE and REE are respectively implemented inside the TrustZone Secure and Non Secure Worlds. For this reason in the remaining part of this paper, Secure World/TEE and Non Secure World/REE are used as synonyms. These two hardware-isolated environments are linked together with a virtualization-enabled implementation of the GlobalPlatform TEE specifications. The virtualization provided by KVM further isolates the user's applications, enabling the use of different operating systems. This eases multitenancy in server and Cloud environments, and enables BYOD paradigm in smart devices. In addition, SELinux isolates in software the virtual machines, protecting guests and

their resources from the other virtual machines and the host itself (e.g., malicious cloud administrators, host privilege escalation exploits, etc.). Lastly libvirt, the main virtualization API used by OpenStack to interact with KVM, takes automatically care of the policy configuration and the tag assignment through its component sVirt.

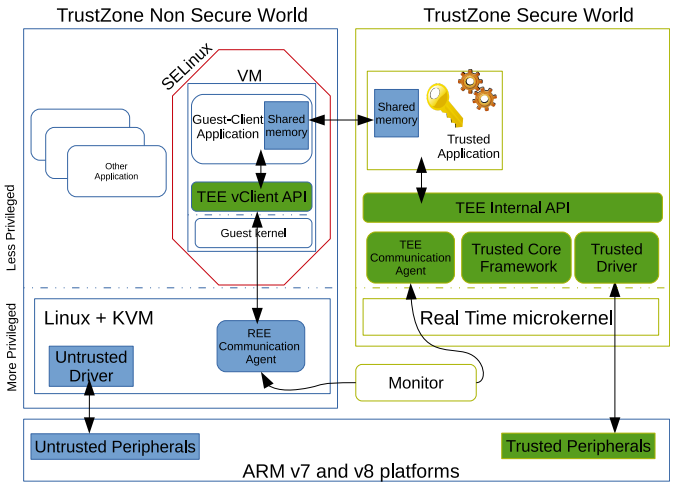

Figure 2. T-KVM architecture, which includes KVM, TrustZone, SELinux and virtualized TEE

A. *Implementation details*

The following part of this manuscript lists the T-KVM implementation challenges and proposes viable solutions.

1) Trusted boot: The first step of the system's chain of trust is performed during the boot procedure. In fact, when the machine boots, the security configuration of the system is not yet in place, and as a result the system is vulnerable to attacks, which target to replace the boot procedure.

In order to minimize this risk, the T-KVM's first stage bootloader is a tiny program stored in a on-chip Read Only Memory (ROM) along with the public key needed for the attestation of the second stage bootloader. Since the first stage bootloader is stored in a read-only memory, it can not be updated and, therefore, it is critical for the security of the system. Soon after the initialization of the key system components, it checks the integrity and boots the second stage bootloader, which is located in an external non-volatile memory (e.g., flash memory). The second stage bootloader then loads the real time microkernel binary in the system's Secure World memory and boots it.

The third stage bootloader of the T-KVM Trusted boot mechanism is a Trusted Application inside the TEE. In fact, when the Secure World OS is up and running, a specific TA checks the integrity of the Non Secure World OS binary (i.e., the Linux kernel) and its bootloader (fourth stage). If this last security check is successful, the fourth stage bootloader runs the Non Secure OS and the system can be considered running. On the other hand, if only one of these checks fails, the boot process will be stopped and the machine will enter in a secure and not operational state. Figure 3 shows the T-KVM Trusted boot chain of Trust.

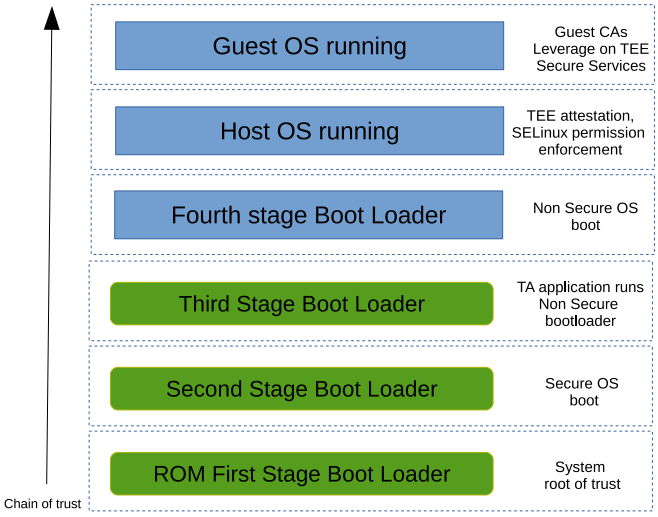

Figure 3. T-KVM Trusted boot procedure

The Trusted boot process is the key element for the attestation of the user space applications because it ensures the integrity of the chain of trust. T-KVM runs in the TEE an attestation service, which is able to check at any moment the integrity of its key components i.e., QEMU, libvirt, the VMs and their resources, etc. libvirt in particular, is extended to attest the VMs identity and integrity in an event-driven manner (e.g., a new VM is booted/shut down, a new device is plugged into the guest, etc.), assuring to users and cloud administrators/providers the authenticity of the workloads run on the hardware. The security assets (fingerprints, keys, etc.) of these binaries are stored in the Secure World and by consequence can be easily updated when necessary.

2) GlobalPlatform TEE support for virtualization: The main novelty introduced by T-KVM is the support for Trusted Computing inside the Virtual Machines. The virtualization of the TEE functions is of utmost importance for the T-KVM architecture, because it links together the applications run in the VMs with the secure/RT services available in the TrustZone Secure World. At the time of writing, the use of the TEE functions in guest operating systems is not included in the GlobalPlatform API Specification. To enable this feature, T-KVM virtualizes the GlobalPlatform TEE APIs, executing the TEE Client API directly in the Guest Operating System. In order to be as much as possible compliant with the GlobalPlatform Specification and to be able to run CAs also at the host level, T-KVM TEE Client API is the only virtualization aware component.

This awareness needs support from the hypervisor infrastructure, for this reason, as depicted in Figure 4, a specific QEMU device is used to implement the TEE control plane and

set up its data plane. All the requests (e.g., initialization/close session, invoke command, etc.) and notification of response are sent to the TEE Device, which delivers them either to the TAs or to the CAs running on the guest OS. To provide good data throughput and latency performance, the data plane is based on a shared memory mechanism. Thus, when a response notification arrives from the TrustZone Secure World, the TEE device notifies with an interrupt its driver, which forwards the related information to the Guest-Client Application. The Guest-CA is now able to read the data from the shared memory, without involving the TEE device in the data transfer.

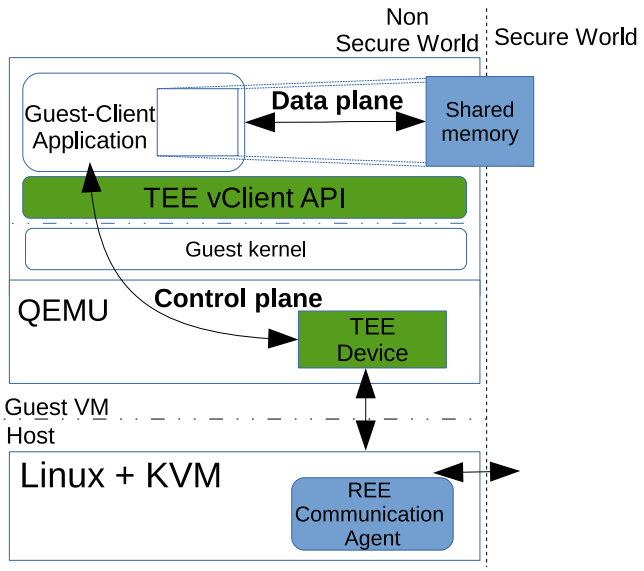

Figure 4. TEE support for Virtual Machines in T-KVM

3) Shared memory: T-KVM needs a zero copy shared memory mechanism to share data between the two TrustZone Worlds and between the virtual machine and the host. The latter in particular is very important in systems where VMs need to communicate with each other frequently (e.g., NFV, HPC, MEC, etc.). Host-guest only shared memory mechanisms which provide high performance and low latency already exist for the KVM hypervisor [21] [22]. What these mechanisms actually lack is the support for TrustZone and, therefore, they are not able to support communication between the TrustZone Non Secure and Secure worlds.

By design, the TrustZone Secure World is able to access the full Non Secure World address space. For this reason, the Trusted Applications are not able to read/write the content of the VMs shared memory unless they know the address where the shared memory area begins. In order to pass this information, the TEE device control plane extends the T-KVM shared memory mechanism, enabling it to send the shared memory address to the Secure World applications. This mechanism need to be secured, especially in the Non Secure World, to prevent attacks and information leakage. T-KVM relies on SELinux to define specific access rules for shared memory and to enforce the shared memory access only to the interested parties.

The encryption of the shared memory area is consciously not considered because, unless hardware accelerators are present in the platform, there would be a performance loss.

4) Secure World: One of the most important parts of the T-KVM architecture is the software running in the TrustZone Secure World. The operating system running in the Secure World should be fast, secure, real-time and free of programming errors (implementation correctness).

The T-KVM architecture empowers the Secure World environment with a real time microkernel. The primary motivation behind microkernels is the small code footprint, which lead to a smaller attack surface and an easier process of formal verification of the code. Moreover, thanks to their real time capability, they enable the deployment of T-KVM in mixed criticality environments such as avionics and automotive. In this context, good candidates are for example seL4, an open-source third-generation microkernel based on L4 and formally verified for functional correctness [23], or FreeRTOS [24] one of the most popular real time operating system for embedded systems.

The microkernel will run in its user space the implementation of the GlobalPlatform APIs, the secure device drivers and the TAs. In order to do this, the Secure World OS does not use the main platform storage device to store files and the security assets of the system. An external, non-volatile memory configured by the Secure World as not accessible by the Non Secure World, is used to this purpose.

Finally, a possible alternative to microkernels is a secure library running in the Secure World such as OPTEE [25]. Despite this solution has a code footprint even smaller than a microkernel, T-KVM uses a microkernel because of its real time features and a higher flexibility for TA developers.

B. Security considerations

Virtual Machines are widely used because of their flexibility and capability to run any operating system. Nonetheless, in order to achieve a higher security level of the system, it is suggested to run single-application operating systems in the T-KVM virtual machines. An example of such an operating system is OSv [26], an open source solution, which is going to be ported to the ARM v8 architecture. For this reason, this work does not discuss the security of the applications inside the virtual machines.

The threat model considered in this paper allows the attacker to completely control one or more virtual machines, both at user and kernel space level. In addition, the cloud administrator, who is permitted to remotely control the virtualized system, is considered as a potential attacker for the identity and integrity of the data.

The security of T-KVM is based on two main assumptions: the attacker does not have physical access to the virtualized system and the first stage bootloader is flawless (thus, the chain of trust is not compromised).

T-KVM has been designed to be compliant with additional hardware accelerators and security modules. For example, SecBus [27] can be used to protect the system against physical attacks on the memory components (e.g., cold boot), Direct Memory Access (DMA) attacks and on-board probing of the external memory bus. Solutions like Network-on-Chip (NoC) Firewall [28], can enhance the compartment isolation granularity at VM level, protecting the system against logical attacks (virus, Trojans) or security vulnerabilities, e.g., corrupted DMA engines.

IV. The T-KVM interrupt management

This section describes how the proposed architecture aims to minimize the interrupt latencies in order to meet real time constraints by providing fast responses.

A. T-KVM interrupt allocation

ARM v7, v8 and earlier architectures have been designed to support two interrupt types: the Fast Interrupt Request (FIQ) for low latency interrupt handling, and the more general Interrupt Request (IRQ), which is commonly available also in other architectures.

The former has higher priority (IRQs are automatically masked by the CPU core when an FIQ arrives) and can directly use some banked registers without the overhead of saving/restoring them through push/pop instructions.

T-KVM takes advantage of the ARM architecture interrupt management design [29], by programming the Secure world to respond only to FIQs and the Normal world to handle IRQs only. This allows critical applications to benefit from fast and high priority interrupts, while isolating them from the Non Secure IRQs.

Lastly, is to be said that the ARM v8 architecture adds the possibility to assign interrupts exclusively to the Monitor (ARM v8 Exception Layer 3). This is an interesting feature for solutions like T-KVM, because it enables the system to isolate interrupts addressed to the Secure World from interrupts addressed to the TrustZone monitor. However, currently available platforms equipped with the Generic Interrupt Controller version 2 (GICv2) such as the ARM JUNO board do not support this feature. T-KVM has been designed to run on existing platforms, and will be adapted and extended when platforms equipped with GICv3 [30] will be available.

The following subsections describe the basic interrupt handling mechanisms for each type of the world execution.

B. Secure World interrupt handling

During the execution of the Secure World, IRQs and FIQs are enabled. FIQs interrupts are handled by the RTOS (red arrow in Figure 5), while IRQs are redirected to the Monitor (green arrow in Figure 5). This minimizes the FIQ interrupts latency during the Secure World execution, because these are directly caught in the handler of the critical application, avoiding any context switch overhead. On the other hand, if an IRQ occurs during the Secure World execution, the Secure OS is preempted and the IRQ is caught in the monitor. At this point, in order to handle the IRQ to the Normal World, the monitor will provoke a context switch by saving the Secure World context and restoring the Normal World, before executing its IRQ handler.

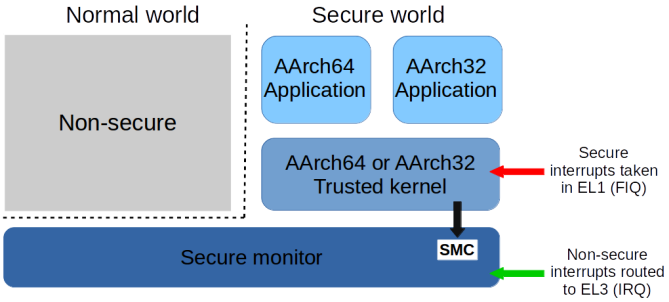

Figure 5. Exception interaction during the Secure world execution

This interrupt management allows IRQs to preempt the Secure world execution when the critical application does not execute a sensitive part (e.g., when it is executing a FIQ handler routine). However, in some specific cases, IRQs can be disabled to prevent the Normal World from interrupting the execution of the Secure World. In that case, the Normal world executes only when there is an explicit request from the Secure world. This is achieved through the Secure Monitor Call instruction.

C. Normal World interrupt handling

Similarly to what happens in the Secure World, both FIQs and IRQs are enabled during the execution of the Normal World: IRQs are directly handled in the Normal World (green arrow in Figure 6) while FIQs are redirected to the TrustZone monitor (red arrow in Figure 6). When a FIQ occurs, the Non Secure OS is immediately preempted in order to propagate the FIQ to the Secure World as soon as possible. This lowers latencies and helps the critical application to meet real time constraints. At this point, the FIQ is caught in the monitor, which executes a world context switch, by saving the Normal World context and restoring the Secure World one, in order to handle the FIQ.

During the FIQ handling into the monitor layer the FIQ mask is enabled and prevents any preemption by another FIQs having higher priority. Then, once the execution control is given to the Secure world, the FIQ management is handled by the trusted application. Therefore, the Secure world could decide to disable the FIQ mask when the critical part is over.

So, once the control is given to the Secure world, the FIQ management is overseen by the FIQ handler of the secure application. Therefore, the Secure world could decide to disable the FIQ mask when the critical part of its FIQ handler is over.

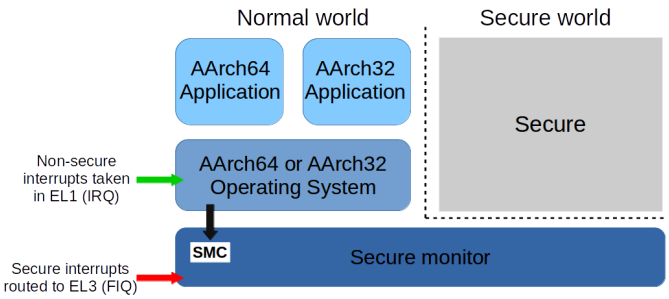

Figure 6. Exception interaction during the Normal world execution

V. Experimental Results

In this section, the result of experimental tests performed on a T-KVM prototype are shown and analyzed. These performance measurements have been performed on an ARMv8 Juno board, which is equipped with two Cortex-A57 and four Cortex-A53 in big.LITTLE configuration. All the workloads for the performance analysis have been executed on the Cortex-A53, which is the default CPU that the platform uses to execute the Secure World OS.

A. SMC propagation

The different isolation layers, which compose T-KVM provide high security, but at a cost of additional overhead. As a matter of fact, a request for a security service from a virtual machine has to pass through the TEE, the host system and SELinux to arrive in the TrustZone Secure World.

For the T-KVM performance analysis of this paper, we focused on the hardware isolation provided by the hypervisor and TrustZone, as the SELinux performance has been measured by the authors in the past [20], and the TEE overhead will be detailed in future works.

For this reason, following the path of a Secure Monitor Call (SMC) from the guest to the Secure World (and then back in the guest) we measured the overhead introduced by T-KVM.

SMC has been added to the ARM instruction set by the ARM Security Extension and it is used to request the execution of a software routine in the TrustZone Secure World passing through the TrustZone Monitor. In the standard KVM implementation, when the SMC instruction is run by a guest OS, its execution is trapped by KVM, which injects an undefined instruction in the guest, forcing it to handle this accordingly. In T-KVM instead, when such instruction is run, the hypervisor traps the guest SMC execution, modifies its arguments and forwards them to the TrustZone Secure World.

In this scenario, two SMC context switches are involved: firstly from Guest to Host, then from Non Secure to Secure World Mode. The overhead assessment of these two context switch operations is the target of the following analysis.

For the first measurement, we implement a bare metal binary blob for KVM, which initializes the Performance Monitoring Unit (PMU) and executes the SMC instruction in the guest. The SMC is then trapped by KVM, which has been modified to immediately return the control to the VM. As soon as the program flow returns back to the guest, it checks the PMU cycle counter status and calculates the overhead. In this way, we are able to measure the overhead of a round-trip context switch between the Guest and the Host when an SMC call is executed.

On the other hand, for the measurements of the context switch overhead between the Non Secure and the Secure Worlds, the PMU cycle counter is set by a Linux kernel module in the host, which executes soon after the world switch request (i.e., the SMC instruction). This provokes an additional switch to the TrustZone Monitor, which saves the Normal World registers, loads the Secure World status and finally jumps to the Secure World. In order to measure the T-KVM context switch cost and not add further overhead, the Secure World immediately runs the SMC call, without executing any meaningful operation. When this instruction is executed in the Secure World, it provokes again a switch to the Monitor Mode, which will now save the secure world context, restore the non secure context, and jump back to the Non Secure World. As soon as the context switch is completed, the Linux kernel reads the PMU cycle counter state and computes the overhead.

The results of both of the above measurements are defined minimal because they are not considering any additional work performed at the destination where they are trapping to. Moreover, in both cases the system frequency is fixed at boot time. However, it is to be considered that, in a real scenario, a trap is followed by the execution of emulation code for a Guest-Host trap, or some secure service for a Non Secure-Secure trap.

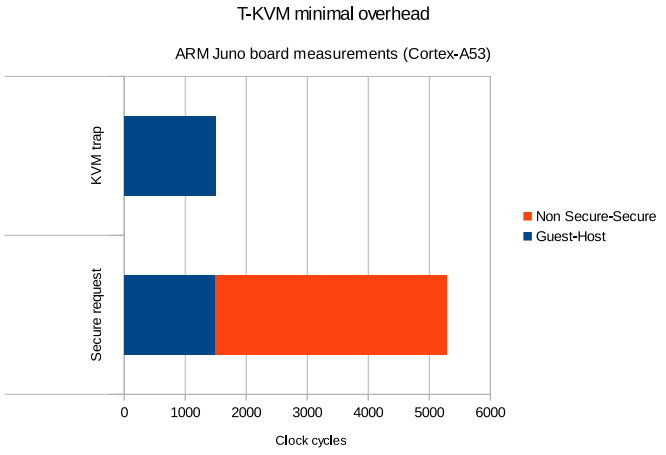

Figure 7. T-KVM overhead measurements

Each test has been repeated five hundred times, running Linux version 3.17 in both the host and the guest environments. ARM Trusted Firmware [31] has been used as firmware infrastructure.

Figure 7 shows that for a Secure request, which is the sum

of the two measurements described before in this section, the cost is in average 5200 clock cycles. This value represents the minimum overhead that a guest system has to pay to request a secure service to the T-KVM Secure World (the Guest-Host minimum overhead is about 1400, while the Non Secure-Secure is about 3700). This has been compared with the minimum overhead that KVM spends to trap the SMC instruction and perform a context switch between the guest and the host, which is what has been described above in the Guest-Host context switch measurement description. This result (in average about 1400 clock cycles) has been measured with the SMC instruction, but it is valid for all the instructions trapped in KVM, as we did not add any specific code to trap the SMC instruction to the standard KVM implementation.

Finally, it is important to notice that the Non Secure-Secure context switch is 2.5 times slower than the Guest-Host. The main reason for this behaviour is in the number of instructions needed to complete the two operations. In particular, the number of registers that the system has to save and restore for the Guest-Host context switch is significantly lower.

B. FIQ latency

As described in the Section III, the T-KVM architecture isolates the time critical applications in the TrustZone Secure World. For this reason, the following performance analysis is focused mainly on the interrupt latencies at the Secure World side, where a RTOS is running.

The proposed architecture, uses FIQ interrupts only for the RTOS, for the reason explained in Section IV. However, the allocation of Fast Interrupts to the RTOS can be implemented in different ways. Hence, following the FIQ path from the vector table to the end of FIQ handler in the Secure world, the overhead introduced by the T-KVM interrupt management has been compared with the performance of different possible implementations:

- **RTOS only** This case represents the T-KVM interrupt management described in Section IV, where FIQs interrupts are handled by the RTOS, while IRQs are redirected to the Monitor.
- **Monitor/RTOS** As above, FIQs are caught in the Monitor. However, in this case, the monitor does not acknowledge the interrupt if the FIQ is not dedicated to the monitor. It forwards FIQs directly to the Secure world, before disabling FIQ trap in the monitor layer (EL3). Therefore, if the FIQ is redirected to the Secure world, the FreeRTOS FIQ handler has to generate an additional SMC at the end of FIQ process in order to return in the monitor to re-enable FIQ trap.
- **Monitor only** The TrustZone monitor (EL3) has full control of the FIQs. FIQs are always trapped in the monitor vector, which can read/write GIC registers to manage the FIQ (activation, acknowledgement, etc.). If the FIQs is not for the Monitor itself, it propagates the FIQ to the Secure world (FreeRTOS).

For all the different FIQ paths described above, a system timer has been programmed to generate periodic interrupts used by the FreeRTOS Tick management to schedule different real-time tasks. Moreover, the FIQ handler installed on the FreeRTOS side, is the same for all tests.

Two different test cases will be taken in consideration for each possible implementation. The first is the measurement of the FIQ latency when the interrupt occurs during the execution of the Normal World (and thus requires a world context switch), and the second is the measurement of the FIQ process time when it occurs during the execution of the Secure world. Moreover, to avoid any cache impact on the results, instruction and data cache memories have been disabled for all the tests.

Moreover, the measurements are split in three parts, each one representing the state of the time at a specific phase:

- ”FIQ trigger” is the time to trig the FIQ in the vector table when the timer reached a specified value.
- ”FIQ propagation in the monitor” is the time for the monitor to execute some operations to manage the FIQ before to redirect it to the Secure world.
- ”FreeRTOS FIQ processing” is the time execution of FIQ handler in FreeRTOS to process the Tick management.

The Timer is set in free-running mode with a frequency of 50 MHz.

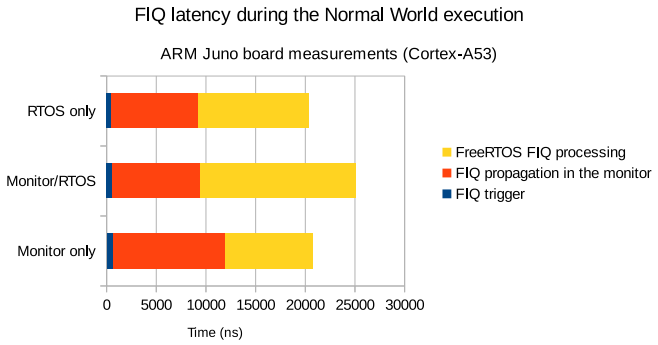

Figure 8. FIQ latency measurements during the Normal world execution

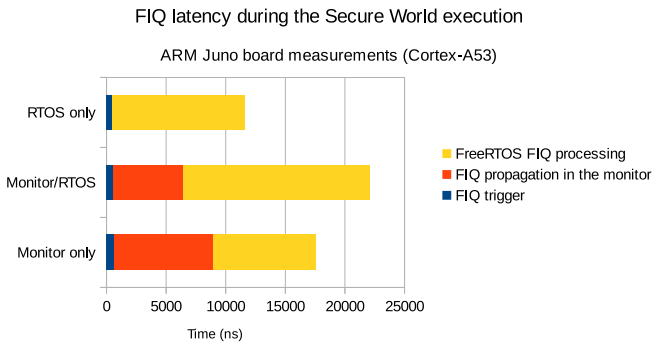

Figure 9. FIQ latency measurements during the Secure world execution

Each test has been repeated one hundred times, running Linux/KVM version 4.0 in the Normal World and a FreeRTOS v8.2.1 in the Secure World. The code needed to run FreeRTOS on the ARM Juno board has been shared by Virtual Open Systems with the FreeRTOS community [32]. Moreover, a modified version of ARM Trusted Firmware [31] developed by Virtual Open Systems, also publicly available as open source software [33], has been used as firmware infrastructure for the monitor layer.

Figure 8 and Figure 9 show that all ”FIQ trigger” parts have approximately the same value for all different tests. This is expected because it corresponds to the hardware time needed to generate the FIQ, and as a consequence it not influenced by the chosen software implementation. Moreover, it is important to notice that the timer value measured to determine the ”FIQ trigger” time has been taken after the execution of few instructions in the vector table handler. So, the value is slightly larger that expected.

About the result of different tests during the execution of the Normal world, Figure 8 shows that the software solution implemented in the Monitor only case has no impact on the total FIQ handling time compared to the basic implementation in the RTOS only. However, the ”FIQ propagation in the monitor” time is more important in the Monitor only case because the monitor executes some instructions, which are normally executed in the FreeRTOS FIQ handler. For the Monitor/RTOS test, there is no overhead during the ”FIQ propagation in the monitor” part because, as the FIQ is dedicated to the Secure world, it forwards directly the FIQ to the FreeRTOS. In this case, an additional communication is implemented to re-enable FIQ trap in EL3 at the end of FreeRTOS FIQ handler and Figure 8 shows that this solution adds an important overhead in the ”FreeRTOS FIQ processing” part.

Additionally, in the case where the FIQ occurs during the execution of the Secure world, Figure 9 shows that the ”FIQ propagation in the monitor” time is lower than the same test realized during the Normal execution. This is expected because, as the world interrupted is the Secure and the monitor redirects the FIQ to the Secure world, the monitor does not execute the context switching operation in this case. Moreover, in the basic interrupt management of the RTOS only test, there is no monitor execution overhead because the FIQ is directly caught in the FreeRTOS FIQ handler.

VI. Related Work

T-KVM is a hypervisor architecture, which mainly targets security and isolation to run virtual machines and real time tasks together on the same hardware platform. Hypervisors' security is a controversial topic in literature. As a matter of fact, solutions like NoHype [34] [35] propose to secure the virtual machines removing the hypervisor, while others use the hypervisor isolation for security applications [16] [18]. In other scientific works, when compared with TrustZone as a security solution, the hypervisor proves a better flexibility, e.g., the Secure World is not able to interpose on and monitor all the important events in the Non-Secure World [17]. The proposed architecture considers the hypervisor as an additional isolation layer, while protecting the security assets through the ARM Security Extensions (TrustZone). T-KVM, combining both solutions and relying on the attestation enabled by the secure boot's chain of trust, is able to provide monitoring features and high security.

In fact, attestation and integrity checks are of paramount importance for the security systems because they allow system designers and administrators to consider a software component as trusted. SecVisor [36], HyperSentry [37] and SPROBES [38] propose different solutions designed for this purpose: the first checks the integrity of commodity OS kernels running the attestation code in hypervisor mode, thus not addressing virtualization. The second enables integrity measurement of a running hypervisor (i.e., XEN) through Intel TxT, hence targeting the x86 architecture. The latter uses TrustZone to enforce kernel code integrity, but without mentioning the attestation challenges in virtualized systems.

These systems are explicitly addressed by solutions such as vTPM [39] or sHype [40], both focusing their efforts on Intel architectures. vTPM proposes a mechanism for the virtualization of TPM functions, which dedicates a VM to route and manage the TPM requests, while sHype integrates the MAC security policy directly inside a typical type 1 hypervisor (i.e., XEN).

On the other hand, the solutions proposed by Narari [41] and Lengyel [42] are designed for ARM devices with virtualization extensions. The first proposes a security architecture with TrustZone, SELinux and virtualization, targeting resource constrained devices. The second combines a hypervisor (XEN) and MAC Security policies (XEN Security Modules), targeting high isolation between VMs but without mentioning TPM access for guest OSes. Both proposals lack of a solution to standardize the access to TPM functions such as the TEE.

VII. Conclusion and Outlook

This paper proposes T-KVM, a new security architecture for ARM v7 and v8 virtualized systems, providing architecture details and a performance analysis. T-KVM's architecture offers strong isolation for guest applications by means of the KVM

hypervisor, ARM TrustZone, SELinux and a virtualization enabled implementation of the GlobalPlatform TEE API. The main contribution of this work is an analysis of the interrupt mechanisms of the ARMv8 architecture with TrustZone, along with a comparison of different possible implementations, which have been described, developed and benchmarked.

The benefits of the T-KVM are its flexibility (programmability of the guests and of the Secure World) and compatibility with the existing Cloud and smart devices architectures (GlobalPlatform and KVM/libvirt support). In fact, since T-KVM adapts and combines existing open source components which are already part of virtualized systems and OpenStack Cloud Computing infrastructure (i.e., KVM, libvirt, qemu, etc.), the support of T-KVM in these environments is straightforward to implement. Lastly, monitoring and (remote) attestation are eased by the combination of a standard hypervisor with TrustZone.

On the other hand, the lack of the ARM VE support in the Secure World does not allow the hardware assisted virtualization of the TEE. Nonetheless, it is possible to functionally implement multiple TEE (or vTPMs) using paravirtualization or virtualization at the application layer.

As for the analysis of the overhead introduced by the proposed solution, it is possible to claim that the performance cost of T-KVM is acceptable. In fact, even if the secure request overhead is significantly higher than a trap in the KVM hypervisor, the execution frequency of the former is expected to be lower than the latter: a service request is issued by a T-KVM guest only to control the communication between the guest and the Secure World, as all the data exchanges will be performed through shared memory. In regard to the FIQ latency tests, the results show that all software solutions to manage the FIQ exclusively handled at EL3 (i.e., Monitor only and Monitor/RTOS cases) introduce overhead which impacts on the FIQ latency. For this reason, in order to privilege the RTOS latency response time, the interrupt management implementation chosen by T-KVM is RTOS only.

Finally, the future work includes the implementation of a complete T-KVM ARMv8 prototype in the direction of automotive and avionics use cases such as infotainment or ADAS. With this in mind a full fledged version of the shared memory mechanism (Section III-A3), GlobalPlatform TEE (Section III-A2) and Trusted Boot (Section III-A1) will be developed, tested and benchmarked. Related to the interrupt management mechanisms, this work has given us the possibility to evaluate different approaches, of which the "RTOS only" has been selected as final solution. Unfortunately, it still lacks of a way to assign interrupts to the Monitor mode. This problem will be investigated in future works, exploring software methods to overcome this limitation of the TrustZone hardware implementation.

Of interest is also the support and integration of T-KVM in OpenStack, which would enable a complete new set of Cloud features based on real time and Trusted Computing services such as real time tasks allocated to the VMs, location aware scheduling of new instances, trusted multitenancy, etc.

ACKNOWLEDGMENT

This research work is an extension of the "T-KVM: A Trusted Architecture for KVM ARM v7 and v8 Virtual Machines" publication [1], awarded best paper at the IARIA Cloud Computing Conference 2015 and supported by the FP7 TRESSCA project (grant number 318036).

This research extension has been supported by the DREAMS project under the grant number 610640.

REFERENCES

[1] M. Paolino, A. Rigo, A. Spyridakis, J. Fanguede, P. Lalov, and D. Raho, "T-KVM: A Trusted Architecture for KVM ARM v7 and v8 Virtual Machines," IARIA Cloud Computing 2015 (CC2015), 2015, pp. 39–45.

[2] M. Wei, B. Heinz, and F. Stumpf, "A Cache Timing Attack on AES in Virtualization Environments," in Financial Cryptography and Data Security, ser. Lecture Notes in Computer Science, A. Keromytis, Ed. Springer Berlin Heidelberg, 2012, vol. 7397, pp. 314–328. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-32946-3_23

[3] B. Saltaformaggio, D. Xu, and X. Zhang, "Busmonitor: A hypervisor-based solution for memory bus covert channels," 2013.

[4] G. Pék, A. Lanzi, A. Srivastava, D. Balzarotti, A. Francillon, and C. Neumann, "On the feasibility of software attacks on commodity virtual machine monitors via direct device assignment," in Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security, ser. ASIA CCS '14. New York, NY, USA: ACM, 2014, pp. 305–316. [Online]. Available: http://doi.acm.org/10.1145/2590296.2590299

[5] N. Elhage, "Virtunoid: A KVM Guest - Host privilege escalation exploit," 2011.

[6] S. E. Madnick and J. J. Donovan, "Application and Analysis of the Virtual Machine Approach to Information System Security and Isolation," in Proceedings of the Workshop on Virtual Computer Systems. New York, NY, USA: ACM, 1973, pp. 210–224. [Online]. Available: http://doi.acm.org/10.1145/800122.803961

[7] M. Paolino, "Isolating ARM platforms: towards secure virtualized embedded systems," February 2015. [Online]. Available: http://www.cspforum.eu/uploads/Csp2014Presentations/Track_8/Strong%20Isolation%20in%20ARM%20Platforms.pdf

[8] G. Chen, "KVM - Open Source Virtualization for the Enterprise and OpenStack Clouds," IDC, 2014.

[9] C. Dall and J. Nieh, "KVM/ARM: Experiences Building the Linux ARM Hypervisor," 2013.

[10] T. Alves and D. Felton, "TrustZone: Integrated hardware and software security," ARM white paper, vol. 3, no. 4, 2004, pp. 18–24.

[11] GlobalPlatform, "TEE Client API Specification, Public Release v1.0," 2010.

[12] Global-Platform, "TEE internal API Specification, Public Release v1.0," 2011.

[13] X. Xu, C. Xiao, C. Gao, and G. Tian, "A study on confidentiality and integrity protection of SELinux," in Networking and Information Technology (ICNIT), 2010 International Conference on, June 2010, pp. 269–273.

[14] A. Jasti, P. Shah, R. Nagaraj, and R. Pendse, "Security in multi-tenancy cloud," in Security Technology (ICCST), 2010 IEEE International Carnahan Conference on, Oct 2010, pp. 35–41.

[15] A. Vahidi and C. Jämthagen, "Secure RPC in Embedded Systems: Evaluation of Some GlobalPlatform Implementation Alternatives," in Proceedings of the Workshop on Embedded Systems Security, ser. WESS '13. New York, NY, USA: ACM, 2013, pp. 4:1–4:7. [Online]. Available: http://doi.acm.org/10.1145/2527317.2527321

[16] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "The Untapped Potential of Trusted Execution Environments on Mobile Devices," Security Privacy, IEEE, vol. 12, no. 4, July 2014, pp. 29–37.

[17] C. Gehrmann, H. Douglas, and D. Nilsson, "Are there good reasons for protecting mobile phones with hypervisors?" in Consumer Communications and Networking Conference (CCNC), 2011 IEEE, Jan 2011, pp. 906–911.

[18] F. Liu, L. Ren, and H. Bai, "Secure-Turtles: Building a Secure Execution Environment for Guest VMs on Turtles System," Journal of Computers, vol. 9, no. 3, 2014, pp. 741–749.

[19] GlobalPlatform, "TEE System Architecture v1.0," 2011.

[20] M. Paolino, M. M. Hamayun, and D. Raho, "A Performance Analysis of ARM Virtual Machines Secured Using SELinux," in Cyber Security and Privacy, ser. Communications in Computer and Information Science, F. Cleary and M. Felici, Eds. Springer International Publishing, 2014, vol. 470, pp. 28–36. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-12574-9_3

[21] C. Macdonell, X. Ke, A. W. Gordon, and P. Lu, "Low-Latency, High-Bandwidth Use Cases for Nahanni/ivshmem," 2011.

[22] M. Paolino, "A Shared Memory zero-copy mechanism for ARM VMs:vosyshmem," February 2015. [Online]. Available: http://www.virtualopensystems.com/en/products/vosyshmem-zerocopy/

[23] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, "seL4: Formal Verification of an OS Kernel," in Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles, ser. SOSP '09. New York, NY, USA: ACM, 2009, pp. 207–220. [Online]. Available: http://doi.acm.org/10.1145/1629575.1629596

[24] R. Barry, "Real time application design using freertos in small embedded systems," 2003.

[25] "OPTEE," February 2015. [Online]. Available: https://github.com/OP-TEE

[26] "OSv," February 2015. [Online]. Available: http://osv.io

[27] J. Brunel, R. Pacalet, S. Ouaarab, and G. Duc, "SecBus, a Software/Hardware Architecture for Securing External Memories," in Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on, April 2014, pp. 277–282.

[28] M. D. Grammatikakis, "System-Level Modeling of a NoC Firewall on Spidergon STNoC," February 2015. [Online]. Available: http://www.cspforum.eu/uploads/Csp2014Presentations/Track_8/System-Level%20Modeling%20of%20a%20NoC%20Firewall%20on%20Spidergon%20STNoC.pdf

[29] "Programmer's Guide for ARMv8-A," March 2015. [Online]. Available: http://infocenter.arm.com/help/topic/com.arm.doc.den0024a/DEN0024A_v8_architecture_PG.pdf

[30] "Programmable Interrupt Controllers: A New Architecture," July 2015. [Online]. Available: http://community.arm.com/groups/processors/blog/2015/07/27/gicv3-architecture

[31] "ARM Trusted Firmware," February 2015. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware

[32] "FreeRTOS v8.2.2 port (AARCH32) for ARMv8 platform (ARM FastModel virtual platform and ARM JUNO Development Platform) using the GCC ARM compiler (arm-none-eabi)," August 2015. [Online]. Available: http://interactive.freertos.org/entries/83649935-FreeRTOS-v8-2-2-port-AARCH32-for-ARMv8-platform-\\ARM-FastModel-virtual-platform-and-ARM-JUNO-Developm

[33] "Add a dispatcher for 32-bit Secure Payload," August 2015. [Online]. Available: https://github.com/ARM-software/arm-trusted-firmware/pull/363/commits

[34] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "NoHype: Virtualized Cloud Infrastructure Without the Virtualization," SIGARCH Comput. Archit. News, vol. 38, no. 3, Jun. 2010, pp. 350–361. [Online]. Available: http://doi.acm.org/10.1145/1816038.1816010

[35] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the Hypervisor Attack Surface for a More Secure Cloud," in Proceedings of the 18th ACM Conference on Computer and Communications Security, ser. CCS '11. New York, NY, USA: ACM, 2011, pp. 401–412. [Online]. Available: http://doi.acm.org/10.1145/2046707.2046754

[36] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes," SIGOPS Oper. Syst. Rev., vol. 41, no. 6, Oct. 2007, pp. 335–350. [Online]. Available: http://doi.acm.org/10.1145/1323293.1294294

[37] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky, "HyperSentry: Enabling Stealthy In-context Measurement of Hypervisor Integrity," in Proceedings of the 17th ACM Conference on Computer and Communications Security, ser. CCS '10. New York, NY, USA: ACM, 2010, pp. 38–49. [Online]. Available: http://doi.acm.org/10.1145/1866307.1866313

[38] X. Ge, H. Vijayakumar, and T. Jaeger, "Sprobes: Enforcing Kernel Code Integrity on the TrustZone Architecture," CoRR, vol. abs/1410.7747, 2014. [Online]. Available: http://arxiv.org/abs/1410.7747

[39] R. Perez and et al., "vTPM: virtualizing the trusted platform module," in Proc. 15th Conf. on USENIX Security Symposium, 2006, pp. 305–320.

[40] R. e. a. Sailer, "sHype: Secure hypervisor approach to trusted virtualized systems," 2005.

[41] H. Nahari, "Trusted secure embedded Linux," in Proceedings of 2007 Linux Symposium, 2007, pp. 79–85.

[42] T. K. Lengyel, T. Kittel, J. Pfoh, and C. Eckert, "Multi-tiered Security Architecture for ARM via the Virtualization and Security Extensions," in Proceedings of the 2014 International Semiconductor Laser Conference, ser. ISLC '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 308–312. [Online]. Available: http://dx.doi.org/10.1109/DEXA.2014.68