2016 IEEE 2nd International Conference on Collaboration and Internet Computing

TrustZone Explained: Architectural Features and Use Cases

Bernard Ngabonziza, Daniel Martin, Anna Bailey, Haehyun Cho and Sarah Martin
Arizona State University
{bngabonz, dlmart11, anna.bailey, hcho67, sarahmartin}@asu.edu

***Abstract*—ARM TrustZone is a hardware security extension technology, which aims to provide secure execution environment by splitting computer resources between two execution worlds, namely normal world and secure world. TrustZone is supported on different flavors of ARM architectures, that include architecture deployed on targets running regular applications, such as mobile devices and architecture for micro-controllers. As ARM is widely deployed on the majority of mobile and micro-controller devices, TrustZone's goal is to provide security for those platforms. In this paper, we will discuss details of different ARM architectures that support TrustZone technology. Then, we will review how TrustZone is implemented in the hardware and software of ARM products. We will also compare TrustZone with other implementations of trusted execution environments on the market.**

I. Introduction

Mobile devices have significantly changed the landscape of computing in recent years by continuously introducing new hardware and software features that were unimaginable before. Since early 2014, the Internet usage on mobile devices has exceeded the usage from PC [1]. A system is usually only secured at the software level. However, a greater level of security can be achieved by building security checks into the hardware of the system. This idea is implemented by the concept of Trusted Execution Environments (TEE).

A widely accepted industry standard of what a TEE is has been defined by GlobalPlatform, as per the GlobalPlatform Compliance Program. It states that "the TEE is a secure area of the main processor" of a device. It must "offer isolated safe execution of authorized security software" [2]. In addition, it should be able to have applications in the areas of user authentication, trusted processing and isolation, transaction validation, usage of secure resources, and certification. TEEs can also be used to protect digital content such as streamed videos from being stolen by keeping all related applications to the secured area of the processor.

In this paper we overview the TrustZone technology on different ARM architectures and discuss the trend of using TrustZone.

II. ARM Architecture Overview

The ARM architecture is a Reduced Instruction Set Computer (RISC) architecture. To date, 8 versions of ARM architectures have been defined, namely ARMv1 through ARMv8. The most popular CPUs in the market now use either the ARMv7 (32-bit, i.e. Cortex-A8, Cortex-A9, Cortex-M4) or ARMv8 (64-bit, i.e. Cortex-A53, Cortex-A57) architecture. Each architecture version may have slightly different profiles. For instance, ARMv7 architecture provides three different profiles: i) ARMv7-A: traditional profile which supports a Virtual Memory System Architecture (VMSA) built around a Memory Management Unit (MMU); ii) ARMv7-R: a real-time profile which supports a Protected Memory System Architecture (PMSA) built around a Memory Protection Unit (MPU); and iii) ARMv7-M: a microcontroller profile which provides low-latency interrupt processing and implements a variant of the ARMv7 PMSA.

The ARM architecture has the following features [3]: i) it has a large uniform register file; ii) data-processing operations only operate on registers, not directly on memory; iii) simple addressing modes; iv) instructions that combine a shift with an arithmetic and logical operation. For example, after the instruction (`ADD R9, R5, R5, LSL #3`) executes, `R5` times `9` is assigned to `R9`; v) auto-increment and auto-decrement addressing modes to optimize program loops. For instance, after the instruction (`LDR R0, [R1, R2, LSR #0x4]!`) executes, `R0` is assigned with the value at the memory address of `R1 + (R2 LSR 0x4)` and `R1` is assigned with `R1 + (R2 LSR 0x4)`. The character `!` indicates a writeback to `R1`; vi) Load and Store Multiple instructions to maximize data throughput. For example, the instruction (`STMFD SP!,{R0-R12, LR}`) is used to store `R0` to `R12` and the return address in `LR` onto the stack and modify the stack pointer accordingly in one instruction, whereas the instruction (`LDMFD sp!,{R0-R12, PC}`) can load registers from the stack and return automatically by overwriting `PC`; vii) conditional execution of many instructions to maximize execution throughput. Most instruction sets only allow branches to be executed conditionally, but most ARM instructions contain a condition field which determines whether the CPU will execute them. For example, the instruction (`ADDEQ R0,R1,R2`) will only execute when the zero flag is set.

In the rest of this section, we discuss the detailed design of ARMv7 and ARMv8 architectures by illustrating processor modes, core registers, system registers, etc.

A. ARMv7-A Architecture

1) Processor Modes: As shown in Table I, an ARMv7 processor has up to 9 different modes depending on if optional extensions have been implemented. The `usr` mode that has a privilege level 0 is where user space programs run at. The `svc` mode that has a privilege level 1 is where most parts of kernel execute at. However, some kernel modules run at special modes instead of `svc`. For example, when a data abort exception happens, a processor switches to the `abt`

978-1-5090-4607-2/16 $31.00 © 2016 IEEE
DOI 10.1109/CIC.2016.63

TABLE I: Processor Modes

Processor Mode	Abbr.	ARMv7 Privilege Level	ARMv8 Exception Level	Security State
User	usr	PL0	EL0	Both
Supervisor	svc	PL1	EL1	Both
System	sys	PL1	PL1	Both
Abort	abt	PL1	EL1	Both
IRQ	irq	PL1	EL1	Both
FIQ	fiq	PL1	EL1	Both
Undefined	und	PL1	EL1	Both
Monitor†	mon	PL1	EL3	Secure only
Hyp‡	hyp	PL2	EL2	Non-secure only

† only implemented with Virtualization Extensions.
‡ only implemented with Security Extensions.

mode automatically. The current processor mode is determined by the mode field (M) of the current program state register (CPSR). Processor mode change can be triggered by exceptions, such as the aforementioned data abort exception. Or, privileged program can directly write CPSR by calling a MSR CPSR_c, #imm instruction, where c stands for the control field that includes processor mode bits and interrupt mask bits.

2) Processor States: If the security extensions (also known as TrustZone) are implemented, a processor has two security states, namely the secure state (s) and the non-secure state (ns). The distinction between the two states is orthogonal to the mode protection based on privilege levels, except that the mon mode is only available in the secure state and the hyp mode that is implemented with virtualization extensions only exists for the non-secure state. The current processor state is determined by the least significant bit of the secure configuration register (SCR) in the CP15 coprocessor. The change of processor states will be discussed in Section III. In the rest of this paper, we use the notation of <mode|state> to represent a mode and state combination.

3) Core Registers: Table II compares the ARMv7 architecture core registers between the application level view and system level view. From the application level perspective, an ARMv7 processor has 14 general-purpose 32-bit registers (R0 to R14), a 32-bit program counter R15 also known as PC, and a 32-bit application program state register (APSR). Two of the 14 general-purpose registers can be used for special purposes: R13 also known as SP is usually used as the stack pointer; R14 also known as LR is usually used to store return address. APSR is an application level alias for CPSR, and it must be only used to access condition flags.

From the system level view, these registers are arranged into several banks, which means a register name is mapped to a collection of different physical registers, governed by the current processor mode. As shown in Table II, each mode except the system mode of the processor has: i) its own banked copy of stack pointer SP; ii) a register that holds a preferred return address for the exception (a banked copy, such as LR_mon, for LP1 modes or a special register ELR_hyp for the hyp mode). Each mode except the usr and system has a banked copy of saved program status register SPSR to save the copy of CPSR made on exception entry. Saving the value of CPSR in banked SPSR registers means the exception handler can: i) immediately restore the CPSR on exception return; ii) examine the value of CPSR when the exception was taken, for example to determine the previous process mode when the exception took place. In addition, the fiq mode has banked copies of R8 to R12. For example, when a processor is executing in the fiq mode, R0 refers to R0_usr, but R12 refers to R12_fiq instead of R12_usr.

Note that processor core registers and program status registers are not banked between the secure state and non-secure state. Therefore, a program can use registers to pass parameters between states. Also, during a processor state switch, a privileged program mostly running in <mon|s> will save the old state' register values and restores the new state's register values. We will discuss more of the state switch in Section III.

4) Coprocessors and System Registers: The ARM architecture supports sixteen coprocessors, namely CP0 - CP15, in which CP15 (System Control coprocessor) is reserved in the architecture for the control and configuration of the processor system. Hardware manufacturer can define other coprocessors for their own purposes.

The system registers in CP15 are categorized in many groups that include i) virtual memory control registers function group (SCTLR, DACR, TTBR0, TTBR1, PRRR); ii) PL1 Fault handling registers; iii) cache maintenance operations; iv) address translation operations; v) security Extensions registers. Given the special purpose of CP15 system registers, many of them are banked between secure and non-secure states. However, the registers that configures the global system status, such as SCR, are not banked. Table III lists some CP15 system registers that are used in this paper.

B. ARMv7-M and ARMv6-M Architectures

With ARMv7, ARM introduced architecture profiles for micro-controller which only support Thumb instruction set. ARMv6-M, which is a sub-set of ARMv7-M, targets ultra low power processor, whereas ARMv7-M targets high performance embedded micro-controllers.

1) Processor Modes: ARMv7-M provides two execution modes (Thread mode and Handler mode). On reset, the process enters in Thread mode. When and exception occurs, the CPU switches to Handler mode. the Code for exception handling runs in handler mode, which is a privilege mode. A return from exception, can go back to the Thread mode. Note that Thread mode may execute at a privilege or an unprivileged Level. Contrary, ARMv6-M can either implement an unprivileged execution or privileged one, but not both at the same time.

2) Cortex-M Registers: Cortex-M processor has 16 core registers (R0 to R15), R0 to R12 are general-purpose registers. It also provides a program counter R15 also known as PC, and a special-purpose program status register (xPSR). R13 also known as SP is usually used as the stack pointer. Cortex-M also offers two banked registers for the stack pointer, SP_main and SP_process. R14 also known as LR is usually used to store return address. APSR an application level program ptatus register used to access condition flags by the application, IPSR is an exception level program status register that holds the exception number, and EPSR, execution program status register, has T-bit to indicate the Thumb execution state.

TABLE II: ARMv7 & ARMv8 Core Registers

AArch32 Application Level View	AArch32 System Level View &\ AArch32 Relationship To AArch64								
	usr	svc	sys	abt	irq	fiq	und	mon†	hyp‡
R0	R0_usr\X0	-	-	-	-	-	-	-	-
R1	R1_usr\X1	-	-	-	-	-	-	-	-
R2	R2_usr\X2	-	-	-	-	-	-	-	-
R3	R3_usr\X3	-	-	-	-	-	-	-	-
R4	R4_usr\X4	-	-	-	-	-	-	-	-
R5	R5_usr\X5	-	-	-	-	-	-	-	-
R6	R6_usr\X6	-	-	-	-	-	-	-	-
R7	R7_usr\X7	-	-	-	-	-	-	-	-
R8	R8_usr\X8	-	-	-	-	R8_fiq\X24	-	-	-
R9 (SB)	R9_usr\X9	-	-	-	-	R9_fiq\X25	-	-	-
R10	R10_usr\X10	-	-	-	-	R10_fiq\X26	-	-	-
R11	R11_usr\X11	-	-	-	-	R11_fiq\X27	-	-	-
R12 (IP)	R12_usr\X12	-	-	-	-	R12_fiq\X28	-	-	-
SP (R13)	SP_usr\X13	SP_svc\X19	-	SP_abt\X21	SP_irq\X17	SP_fiq\X29	SP_und\X23	SP_mon\N/A	SP_hyp\X15
LR (R14)	LR_usr\X14	LR_svc\X18	-	LR_abt\X20	LR_irq\X16	LR_fiq\X30	LR_und\X22	LR_mon\N/A	-
PC (R15)	PC	-	-	-	-	-	-	-	-
APSR	CPSR	-	-	-	-	-	-	-	-
N/A	N/A	SPSR_svc \SPSR_EL1	N/A	SPSR_abt	SPSR_irq	SPSR_fiq	SPSR_und	SPSR_mon \SPSR_EL3	SPSR_hyp \SPSR_EL2
N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	ELR_hyp \ELR_EL2

† only implemented with Security Extensions. ‡ only implemented with Virtualization Extensions. A cell filled with - means the user mode register is used when referred. A cell filled with a register name means the register is banked. A cell filled with N/A means no register is available for this mode or level view.

TABLE III: Some CP15 System Registers on ARMv7

Register	Name	Security State	Remarks
VBAR	Vector Base Address Register	Banked in both states	
MVBAR	Monitor Vector Base Address Register	Secure state, monitor mode	
ISR	Interrupt Status Register		
SCR	Secure Configuration Register	NS bit	Only with Security Extensions
TTBRx	Translation Table Base Register (0), (1)	Banked in both states	
TTBCR	Translation Table Base Control Register	Banked in both states	
DACR	Domain Access Control Register		
SCTLR	System Control Register	Banked in both states	
NSACR	Non-Secure Access Control Register	Security Extensions	
SDER	Secure Debug Enable Register		

C. ARMv8-A Architecture

1) Execution States: ARMv8 provides two states of execution, AArch32 that uses 32-bit registers, also compatible with ARMv7 architecture and AArch64 which uses 64-bit registers, also having a 64-bit memory address space (virtual and physical). Coprocessors's registers are 32-bit wide for both states.A 64-bit operating system can host both 32-bit applications and 64-bit applications but a 32-bit OS only hosts 32-bit apps. Moreover, on a 64-bit OS, one application cannot have mixed instructions (32-bit instructions mixed with 64-bit ones). When running a 32-bit application on the top of a 64-bit operating system and an interrupt occurs, the system must change to AArch64 to handle the interrupt, the switch from AArch32 can also be done through a Supervisor Call SVC. For both cases a return from an exception will cause a switch back to AArch32.

2) Exception Levels: AArch64 adds a new exception level EL3 (ARMv8 uses the term exception level, while ARMv7 uses the term privilege level but both means the same thing). In ARMv8 we have four levels of privilege: EL0, EL1, EL2 and EL3. AArch64's firmware and secure monitor run at EL3, contrarily to ARMv7's monitor mode which runs at PL1. To ensure compatibility with ARMv7 when running a 32-bit system (AArch32); all secure-world privileged levels are at EL3. Otherwise, when the <mon|s> code is 64-bit, an AArch32 code doesn't have access to EL3 but can run at both EL1 and EL0

3) Core Registers: An ARMv7 processor has 31 general-purpose registers (X0 to X30). Table II also compares registers relationship between ARMv7 and ARMv8. From the table we can see that there are no ARMv7's monitor mode LR_mon and SP_mon mapping to ARMv8; that is because for ARMv8, monitor mode only exists in the 64-bit execution state and is at a level of privilege, EL3, which does not exist in ARMv7. The equivalents of stack point and link register for ARMv8 are SP_EL3 and ELR_EL3 respectively, which are not general purpose registers, but special registers for EL3.

III. TrustZone: The ARM Security Extensions

As mentioned previously, TrustZone [4] is an optional hardware security extension of the ARM processor architecture, which includes bus fabric and system peripherals. The security of TrustZone is based on the idea of partitioning all of the System on Chip (SoC)'s hardware and software into two worlds: secure world and normal world. The secure world is everything runs when the processor state is secure, and normal world is everything runs when the processor is in the non-secure state. Hardware barriers are established to prevent normal world components from accessing secure world resources; the secure world is not restricted. Specifically, the memory system prevents the normal world from accessing i) regions of the physical memory designated as secure; ii) system controls that apply to the secure world; and iii) state switching outside of a small number of approved mechanisms.

This partitioning may be physical and/or virtual. For instance, a physical processor core is shared by the normal and secure world in a time-sliced fashion, which gives both worlds the illusion that it owns the processor. The secure world enables the construction of an isolated programmable environment that can run a wide range of security applications.

A. TrustZone Hardware Architecture: Processor and Others

ARM has implemented this split-environment processor with various system IP additions. These unique components are used to enforce security restrictions while preserving the low power consumption and other advantages of ARM's designs. Some of the features are described in the specification for ARM's Advanced Microcontroller Bus Architecture version 3 (AMBA3).

The AMBA3 AXI to APB Bridge allows for secure communication between a CPU and peripherals. The Advanced eXtensble Interface (AXI) bus, which is the main system bus, contains an active-high non-secure (NS) bit that indicates whether a read/write operation is directed to secure or non-secure memory. The Advanced Peripheral Bus (APB), whose low bandwidth reduces power consumption, connects to the AXI bus via a bridge. As the APB does not check for security due to backward-compatibility concerns, the bridge checks for appropriate permissions and blocks unauthorized requests.

Like the AXI to APB bridge, the Cache Controller also looks for an NS bit. This bit is basically treated like a 33rd address bit: the first 32 bits provide the location, and the NS bit indicates which world it refers to. Since both worlds share the same physical cache, the same location may have two distinct addresses, requiring a controller to look up the correct location. This also includes L2 cache and other smaller locations.

The Direct Memory Access (DMA) Controller is used to transfer data to physical memory locations instead of devoting processor cycles to this task. This controller, which uses AXI, can handle Secure and Non-secure events simultaneously, with full support for interrupts and peripherals. It prevents non-secure access of secure memory.

The TrustZone Address Space Controller (TZASC) allows dynamic classification of AXI slave memory-mapped devices as secure or non-secure. Controlled by the secure world, the TZASC allows partitioning of a single memory unit rather than requiring separate secure and non-secure units. The TZASC allows an arbitrary number of partitions to be created.

The TrustZone Memory Adapter (TZMA) allows division of on-chip static memory into secure and non-secure halves. The halves must be in multiples of 4 kB, and the total unit cannot be larger than 2MB. The TZMA can be controlled using the R0SIZE input signal.

The Generic Interrupt Controller (GIC) is a device that handles secure and non-secure prioritized interrupts. It prevents non-secure interrupts from unauthorized access. To prevent denial-of-service attacks, the GIC only handles interrupts in the lower half of the priority hierarchy.

Finally, the TrustZone Protection Controller (TZPC) is a signal-control unit. It has three 2-bit registers to control up to 8 signals; one of these registers, TZPCR0SIZE, can communicate with the TZMA.

B. TrustZone Software Architecture

The TrustZone Hardware, where hardware extensions enforce a separation of secure and non-secure software, is more resource-efficient than the use of two separate processors. The secure world can be implemented as a full-fledged operating system, a software library, or somewhere in between. While the library design is simple, it is only suitable for occasional secure calls. Thus, the operating system model will be the focus of this discussion.

When using TrustZone to implement concurrent secure and non-secure operating systems, a general operating system, such as Linux, Android, etc., would run in the normal world and a security subsystem, such as OP-TEE or a customized Linux [5], would run in the secure world. The functionality of the device can be described in two scenarios: first, to properly boot both operating systems; second, to provide a proper communication framework between the two.

Securely booting both parts of this system is a key concern. Without proper verification of both images, the device may inadvertently boot a malicious version, giving attackers an entry route. Therefore, ARM has designed TrustZone-enabled systems to use a Secure Boot Sequence. To build a chain of trust, each step can be cryptographically verified, usually starting with a vendor-specific public key. One-Time Programmable (OTP) techniques can be used to generate unique keys for each part of the device. As a first step, a ROM-based bootloader initializes important peripherals, such as memory units. Next, the secure world is allowed to boot from a flash device; if the CP15SDISABLE register has been cleared by the ROM initially, the secure world will set it after modifying its settings. This value locks the configuration of the secure world, such that the settings are made immutable until the next reboot.

Once the device has finished booting, the two operating systems can communicate with each other through the "monitor" kernel mode, which acts like an ordinary context switch. Normal world software can only enter this mode by using a hardware interrupt, an external abort signal, or the software instruction `SMC`. The hardware interrupts and aborts are asynchronous and only support a full world switch, while `SMC` also supports message passing without a complete

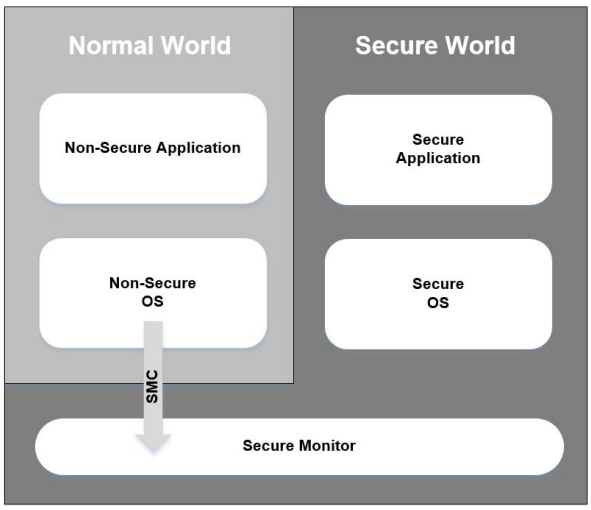

Fig. 1: TrustZone on ARM Cortex-A

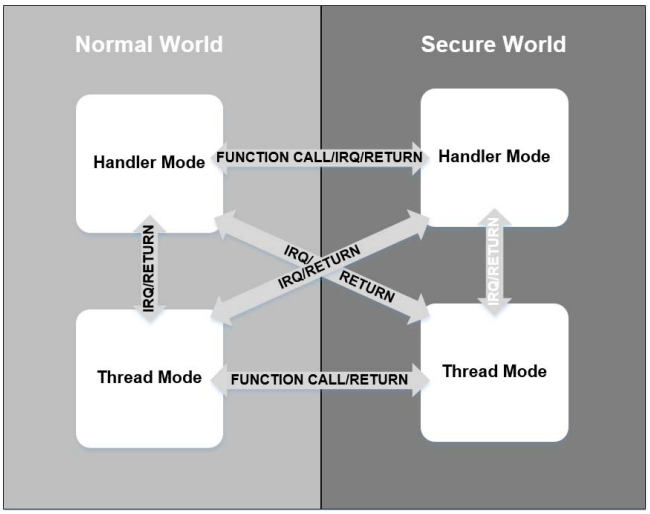

Fig. 2: TrustZone on ARM Cortex-M

changeover. Secure world software can use these methods, or write directly to the CPSR. Monitor mode executes with interrupts disabled due to volatility concerns. In processors that include VFP and NEON hardware units, ”lazy” context switching is supported to reduce overhead.

ARM has made specific recommendations for using interrupts in context switches. IRQ is the recommended source for the Normal world, while FIQ is recommended for the secure world. The Secure Control Register (SCR) indicates which signal to watch for. If core is already in the correct world, the signal is discarded; else, the hardware will enter monitor mode. To prevent tampering with this convention, FIQ is controlled by bits in the CPSR, which can only be modified by secure world software. There are no such restrictions on the use of IRQ. There are three vector tables for interrupt handling: secure world, normal world, and monitor mode. In multiprocessor systems, the secure world can either run on all processors, or restrict itself to one core. If it is restricted, then only the normal world on the shared core should have the capability to switch to the secure world; all other cores will ignore such signals, and can use the FIQ for other purposes.

To facilitate usage of the TrustZone software at an application level, ARM has also introduced public specifications for a TrustZone API (TZAPI). TZAPI is primarily concerned with establishing a standard interface for sending information to and from the secure world, regardless of the style of secure-world implementation. It also makes use of the idea of ”World-shared” memory for increased efficiency.

C. ARMv8-M TrustZone

ARMv8-M integrates TrustZone technology into cortex-M. Also like ARMv7-M, ARMv8-M comes in two profile too, ARMv8-M Baseline and ARMv8-M Mainline; similar to ARMv6-M and ARMv7-M respectively, but with some important enhancements. Like for cortex-A TrustZone is an optional hardware security extension on an ARMv8-M processor; which provides additional security states, Secure World and Normal World, with Secure World able to access all resources, but Normal World only accessing resources it has been allocated to. Also on cortex-M, TrustZone secure and non-secure states are orthogonal to Thread and Handler modes mentioned previously, but Unlike cortex-A, any ARMv8-M's mode (thread or handler) can be privilege or unprivileged.

ARMv8-M TrustZone doesn't provide monitor mode, which improves interrupt latency, there is no need to go through an additional transitional mode. As mentioned previously, Cortex-A's security state is determined by the value of the NS-bit on the secure configuration register. However, the process security state on ARMv8-M is determined by weather the code being executed resides in a secure memory or a non-secure one. ARMv8-M TrustZone's security state changes when the execution switches to the code in a memory region which belongs to a different security domain. This means non-secure applications can directly call secure applications (running in thread or handler mode), and vice-versa.

IV. Comparing TrustZone with Other Hardware-based Security Solutions

TrustZone can be compared to other TEE technologies using GlobalPlatform's standards. In addition, Sabat et al. [6] provide several categories of criteria for comparison: functional, security, and deployability. Functional criteria include Protected Execution, Sealed Storage, Protected Input, Protected Output, and Attestation. Overall, these criteria measure the physical protection of the data through the entire usage cycle: receipt, input, processing, output, and validation. Security criteria include Data Separation, Information Flow Control, Sanitization, Damage Limitation. This category focuses more on the mechanisms to avoid specific attacks on the system. Finally, Deployability criteria measure the barriers to adoption; these include Support of Legacy Systems, Cost, Overhead, and SEE Performance. As there is some overlapping between the Functional and Security criteria, we will attempt to focus on salient points.

A. Comparing TrustZone to Secure Element/TPM

According to Global Platform, a secure Element is a tamper-resistant separate platform, in which secure applications and their cryptographic data are stored. Compared to TrustZone, a secure element has less computational capability as it run on separate hardware and does not have access to main system's cpus. TrustZone's trusted execution environment runs on the same cpu as the rich operating system.

On the other hand a secure element guarantees a higher level of security by providing a secure key storage and a root of trust. TrustZone doesn't provide security for non-volatile storage and lacks a guaranteed root of trusts.

While Trustzone and secure element are mainly used for Mobile devices, Trusted Platform Modules are mainly deployed on Notebook PCs. Same as secure element, TPMs provide a root of trust for cryptographic key storage, also has low performance due to a slow cpu bus and doesn't leverage the main CPU. Moreover, secure element and TPM can be integrated with to TrustZone, thus providing a root of trust and bootstrap trust.

B. Intel TXT and SGX

Intel Trusted Execution Technology (TXT) is a virtual machine protection mechanism through a combination of hardware extensions to CPU and supporting software. Hardware extensions compromise virtual machine extension (VMX), which permits to create separate guests virtual machine (VM), and safer machine extension (SMX), responsible for the measurement of virtual machine monitor.

Intel TXT protect physical page of a virtual machine from being accessed by other virtual machines. In addition, it ensures secure communication between I/O devices and the protected virtual machine.

Intel TXT [7] is designed to rely on TPM and provide a trusted way to load and execute system software, such as operating system or virtual machine monitor. TPM allow Intel TXT to provide protection for long-term storage. However, this also leads to a greater reliance on the manufacturer's trustworthiness than with TrustZone. Such reliance can be problematic [8].

Both Intel TXT and TrustZone provide the ability to run trusted and untrusted Operating System side by side by partitioning memory between them; this can result in a scarcity of resources, especially for the trusted execution environment.

Intel Software Guard eXtension (SGX) is a hardware extension on Intel CPU's which enables the creation of secure container for data and code using that data. Both trusted and and untrusted execution have access to all system resource which provide security and high performance at same time.

C. TI M-Shield and AMD

An interesting issue to note is that TrustZone has been adopted by other manufacturers. While Intel pursues SGX, AMD and Texas Instruments have both decided to integrate TrustZone technology in their products. AMD uses TrustZone in its 64-bit Advanced Processing Units(APU's). TI integrated TrustZone into its M-Shield Security framework used in the OMAP2 and OMAP3 processors [9].

D. Compare TrustZone to Virtualization

Machine virtualization, using hypervisor, is a full system virtualization of all hardware resource which include memory, cpu, devices and also entire ISA(Instruction Set Architecture. The hypervisor multiplex access to real resources between various virtual machine. TrustZone only multiplex cpu between a normal world and secure through the monitor mode; other resources such as memory are physically split among them. TrustZone can be used as a virtualization tool, capable only of hosting two operating system, in this the monitor mode process will server as a virtual machine monitor (VMM).

Both Hypervisor and TrustZone can be used for out of VM kernel monitoring. For this use, secure world and the Hypervisor introspect normal world and hosted VM respectively. The main difference between TrustZone kernel monitoring and Hypervisor VM monitoring is that the hypervisor run at the same level of privilege as the hosting machine. If there is a bug in the hypervisor the hosted can exploit, the hosting machine would also be compromise, contrary to Trustzone where the monitoring code runs in a isolated environment (secure world) and even a normal world is unsafe, it still has no access to code running in secure world.

E. Evaluation of TrustZone

The trust of a computation system is usually bootstrapped from some foundational *root of trust*, that is typically the secrecy of a private key [10, 11]. And, the secrecy of such a private key is guaranteed by some physical features of the embedded devices. Even though being marketed as 'TrustZone', this technology does not directly provide secure key storage. Therefore, it does not provide any kind of *root of trust* but just a system-wide isolation of two execution environments. As a result, another physical device has to be introduced to work with TrustZone to provide a root of trust and bootstrap trust. With hardware-based TPM or MTM, trust can be bootstrapped from the hardward roots of trust. However, TrustZone itself does not provide such hardware roots of trust [5].

V. Research Projects that use TrustZone

A DACR-based approach, namely ARMLock [12], was proposed to isolate untrusted modules in one host application. ARMLock supports up-to 13 simultaneously running domains for each process, since Linux uses domain 0, 1, and 2 for the kernel, user-space, and, device memory respectively. ARMLock assigns unique domain IDs to untrusted modules, and updates DACR when entering and leaving a sandbox so that the untrusted code can only access the current running domain. System calls are devised for cross-domain communications. In ARMLock, the kernel is trusted.

TrustZone is also used to simulate virtualization techniques. Pinto et al. [13] proposed to run a general purpose OS in the normal world and a real-time OS in the secure world. They designed the VMM to run in the monitor mode. They assigned FIQ to secure world and IRQ to normal world. They did not explain how to prevent the secure world from accessing the normal world. In their design, either OS can not preempt the other OS. So starvation is normal if one of the OSes does not yield its control of CPU.

TrustZone provides the privileged `smc` instruction to switch processor state from normal world to secure world, and vice versa. Therefore, any normal world code in a privileged mode can call `smc`, and secure world monitor will be triggered. To control this access, SeCRet [14] proposed authentication of normal world processes.

In the application area, TEE's can be used as part of a multi-factor authentication protocol. Such protocols, such as Oauth and FIDO, allow users to authenticate themselves using an additional device besides a standard text password. TPM has already been used in the OpenID protocol [15]. However, TrustZone allows for faster adoption due to the popularity of ARM-platform smartphones and other mobile devices; any sensitive data like encryption keys can be handled by the secure world. As a member of the FIDO Alliance, ARM has issued a whitepaper with a suggested architecture of the FIDO protocol using TrustZone [16], but a real-life implementation has not been constructed to date.

VI. Conclusion

Overall, one can see that ARM's TrustZone is a viable implementation of the Trusted Execution Environment. ARM has provided both hardware and software mechanisms to enforce the separation of secure and unsecure data. The hardware controllers provide an acceptable level of backward compatibility, and try to avoid excessive reductions in latency. The software architecture allows flexibility: the secure world can be a slave process, a full operating system, or somewhere in between. The full operating system model, at least in single-core machines, is well implemented. Compared to the TPM approach, espoused by Intel, TrustZone is more economical with hardware, and avoids the scenario of an all-powerful "black box" being the ultimate controller of the system. While TrustZone does not allow as much control over peripherals as a hypervisor, it has less of an attack surface.

Acknowledgment

This work was supported in part by grants from the Center for Cybersecurity and Digital Forensics at Arizona State University.

References

[1] Rebecca Murtagh. Mobile now exceeds pc: The biggest shift since the internet began. http://searchenginewatch.com/sew/opinion/2353616/mobile-now-exceeds-pc-the-biggest-shift-since-the-internet-began, 2014.

[2] GlobalPlatform. GlobalPlatform made simple guide: Trusted Execution Environment (TEE) Guide. http://www.globalplatform.org/mediaguidetee.asp, 2016.

[3] Arm architecture reference manual: Armv7-a and armv7-r edition. 2012.

[4] Arm security technology building a secure system using trustzone technology. 2009.

[5] Johannes Winter. Trusted computing building blocks for embedded linux-based arm trustzone platforms. In *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, pages 21–30. ACM, 2008.

[6] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. The dual-execution-environment approach: Analysis and comparative evaluation. In *ICT Systems Security and Privacy Protection*, pages 557–570. Springer, 2015.

[7] David Grawrock. The intel safer computing initiative, 2005.

[8] Joanna Rutkowska. Intel x86 considered harmful. Technical report, Invisible Things Lab, 2015.

[9] Jerome Azema and Gilles Fayad. M-Shield Mobile Security Technology: making wireless secure. Technical report, Texas Instruments Incorporated, 2008.

[10] Bryan Parno, Jonathan M McCune, and Adrian Perrig. Bootstrapping trust in commodity computers. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 414–429. IEEE, 2010.

[11] Bryan Parno, Jonathan M McCune, and Adrian Perrig. *Bootstrapping Trust in Modern Computers*, volume 10. Springer, 2011.

[12] Yajin Zhou, Xiaoguang Wang, Yue Chen, and Zhi Wang. Armlock: Hardware-based fault isolation for arm. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 558–569. ACM, 2014.

[13] Sandro Pinto, Daniel Oliveira, Jorge Pereira, Nuno Cardoso, Mongkol Ekpanyapong, Jorge Cabral, and Adriano Tavares. Towards a lightweight embedded virtualization architecture exploiting arm trustzone. In *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*, pages 1–4. IEEE, 2014.

[14] Jinsoo Jang, Sunjune Kong, Minsu Kim, Daegyeong Kim, and Brent Byunghoon Kang. Secret: Secure channel between rich execution environment and trusted execution environment. In *Proceedings of NDSS*, 2015.

[15] Andreas Leicher, Andreas U. Schmidt, Yogendra Shah, and Inhyok Cha. Trusted Computing Enhanced OpenID. In *2011 International Conference on Internet Technology and Secured Transactions (ICITST)*, pages 1–8, 2010.

[16] Rob Coombs. Securing the Future of Authentication with ARM TrustZone-based Trusted Execution Environment and Fast Identity Online (FIDO). Technical report, ARM Limited, 2015.