Hindawi
Security and Communication Networks
Volume 2023, Article ID 7369634, 19 pages
https://doi.org/10.1155/2023/7369634

Review Article

Investigating TrustZone: A Comprehensive Analysis

Qinyu Zhu,[1] **Quan Chen**,[1] **Yichen Liu**,[1] **Zahid Akhtar**,[2] **and Kamran Siddique**[1,3]

[1]*Department of Information and Communication Technology, School of Computing and Data Science, Xiamen University Malaysia, Sepang 43900, Malaysia*
[2]*Department of Network and Computer Security, State University of New York Polytechnic Institute, Utica, NY 13502, USA*
[3]*Department of Computer Science and Engineering, University of Alaska Anchorage, Anchorage, AK 99508, USA*

Correspondence should be addressed to Kamran Siddique; ksiddique@alaska.edu

Received 8 May 2022; Revised 19 December 2022; Accepted 22 December 2022; Published 14 April 2023

Academic Editor: Saed Alrabaee

The advent of the Internet and portable devices, including smartphones and watches, has brought unprecedented opportunities for embedded application systems developments. Along with these developments, there is an increasing need for embedded devices to handle important services, such as the ability to pay bills or manage bank accounts remotely via mobile phones. Such applications and developments have also highlighted the issues of cyberattacks and computing network security--these developments have made mobile phones a potential target for malware, trojans, and viruses, so it is critical to design a set of security technologies for embedded devices. In fact, security has become an essential requirement in the process of embedded system design. Thus, ARM has proposed system-level security solutions based on TrustZone technology. TrustZone technology is tightly integrated with Cortex™-A processors and extends the system through the AMBA® AXI bus and specific TrustZone system IP blocks to protect peripherals such as secure memory, encryption blocks, keyboards, and screens from software attacks. It divides the system into TEE (Trusted Execution Environment) and REE (Rich Execution Environment) by hardware and provides intrinsic software security services and interfaces. More precisely, it has built system security by combining hardware and software. It is worth noting that it does not influence performance, power consumption, and area as much as possible. Owing to such characteristics, the technology has gained the wide attention of researchers worldwide. There is lack of systematic documentation of the technology. Therefore, this paper documents the significant progress achieved in the field. In particular, this article mainly analyses the primary mechanism implementation, and how to build the Trusted Execution Environment in different environments. Then, this paper discusses the related research works in the academic field and business applications of the technology. Furthermore, the advantages and weaknesses of the TrustZone technology as well as the proposed possible solutions aiming at the deficiency are outlined. Finally, a comparison of TrustZone technology with another mainstream commercial SGX, and future directions are presented.

1. Introduction

Recently, due to the rapid development of mobile Internet technology, people are more frequently using mobile devices in daily life. In this situation, the development of embedded systems places a critical position. With the development and deployment of various Internet applications, the use and facility requirement of the embedded system becomes more complex. People can use embedded devices to make purchases, to make reservations, and even now to make large transactions, and to manage bank accounts has become very common, so protecting sensitive private data and its operation processes in an adverse environment have become a hot research point. In [1, 2], some approaches to the cybersecurity protection of basic networking facilities are proposed. Thus, developing a set of security solutions for embedded systems becomes more urgent together with their deployments. In this security solution, design should be avoided to add conditions and components to supply a gap several times. This can avoid the embedded system not being too complex on the one hand. On the other hand, it can prevent more and more complicated designs. Meanwhile, on account of various security threats on the Internet, when devices with embedded systems connect to the Internet, it

should be ensured that the devices can perform normal operations and protect sensitive data and processes under a malicious attack and exploitation. This design should cooperate mutually with hardware and software because pure hardware design will lead to an increase in power, bigger cost, and being weak to newly rising security problems. Conversely, refined software design will lead to (I) an increase in system complexity, (II) decline in operating efficiency, and (III) unable to avoid the security problem that is being demolished [3]. In 2002, embedded processor IP supplier ARM corporation came up with TrustZone technology to provide a comprehensive system security solution, which is a system-wide security approach targeting a wide range of applications on high-performance computing platforms, including secure payments, digital rights management (DRM), enterprise services, and Web-based services [4]. It achieved security through only minor design tweaks in hardware. In 2009, Apple adopted TrustZone to protect its Touch ID data in iPhone 5s [4]. It is worth noting that even when IOS is fully compromised, the Touch ID data will be still safe through this technique. From that, lots of Major vendors started to use TrustZone technology. In 2017, Google coerced TEE as a necessary component on any Android device with a fingerprint scanner [4]. As the extension support of TEE in the ARM platform, TrustZone divides the resources into two parts: Trusted Execution environments (TEEs) and Rich Execution Environment (REE). TEE gives applications a Trusted Execution Environment to store sensitive data and process them. REE is the execution environment for normal applications. TEE and REE are isolated by hardware which guarantees the effectiveness of security. Meanwhile, TrustZone also divides applications into two parts: TA (Trusted Application) and CA (Client Application) [5]. TA runs in TEE, containing storage and execution of sensitive data. CA only runs in REE and can invoke sensitive data only by calling TA applications.

In a word, TrustZone is an excellent technology. Most security-related functions on embedded devices are implemented based on TrustZone, including runtime security, secure storage, secure computing, private key signature, fingerprint comparison, authentication, authorization management, and DRM authentication.

The remainder of this article is organized as follows. Section 2 presents an analysis of the basic system architecture of TrustZone and its environment. Section 3 discusses build Trusted Execution Environment (TEE) based on TrustZone. The analysis of research works based on TrustZone is outlined in Section 4. Whereas advantages and disadvantages of TrustZone security technology are explored in Section 5. The security performance optimization is provided in Section 6. A comparative analysis between ARM TrustZone and SGX is explained in Section 7. Future directions are discussed in Section 8. Finally, the conclusion is drawn in Section 9.

2. TrustZone Security Technology Architecture Overview

In order to introduce the architecture of TrustZone technology more clearly and completely, this article will analyze the hardware architecture, software architecture, and Implementation of the TZ security mechanism in detail.

2.1. Hardware Architecture. The core idea of TrustZone's hardware architecture is isolation. These hardware isolations include interrupting isolation, on-chip, off-chip RAM and ROM isolation, peripheral hardware isolation, and external RAM and ROM isolation. To realize various isolation at the hardware level, the hardware and processor need to be expanded accordingly.

First, one aspect of the TrustZone hardware architecture is implementing hardware expansion in the CPU. Figure 1 shows the TrustZone hardware architecture, which isolates the existing core of the CPU and divides the CPU core into the security world and the normal world [6]. The physical sense of a single processor core is composed of two virtual processor cores: the security processor core and the normal processor core. Therefore, a single processor core in the physical sense can use its isolated two generated virtual cores to execute programs in the normal world and the security world, respectively, and simultaneously. This isolation operation can ensure that secure memory and peripherals can deny nonsecure processes by hiding them in the operating system to achieve hidden security defense. In addition, virtual isolation also eliminates the need to design a particular CPU security core. Thus, it allows the CPU's normal world operating environment and security protection software to run simultaneously and saving chip size, power consumption, and design and manufacturing costs.

To manage and switch the state of the two processors, TrustZone introduces a unique mechanism-the monitoring mode. The primary function of this mode is similar to the context switching function on the traditional operating system, i.e., ensuring that the processor can safely and accurately save its working environment before switching and correctly restoring the system operation in the changing climate. Entering the monitoring mode from the normal state, requesting to switch to the security state in the normal state, is strictly limited by TrustZone. It can only be entered through an interrupt, external interrupt, or direct call of SMC instruction. There are no strict restrictions on entering the monitoring mode from the security state. Insecurity state can switch to normal way through the exception handling mechanism or directly override the Current Program Status Register.

The second aspect of TrustZone hardware architecture is the hardware expansion in memory. To separate all the hardware and software resources of SOC into two parts, the TrustZone extends the hardware in memory. Such separation isolates the memory and divides it into two parts: the normal world for storing all other content resources and the secure world for storing security systems. The CPU has independent physical address Spaces when running processes in Trusted Execution Environment (TEE) and Rich Execution Environment (REE) [7]. Under TEE, the CPU can access resources in the corresponding address space of the two environments. While under REE, the CPU can only access its own address space. Therefore, when a process is

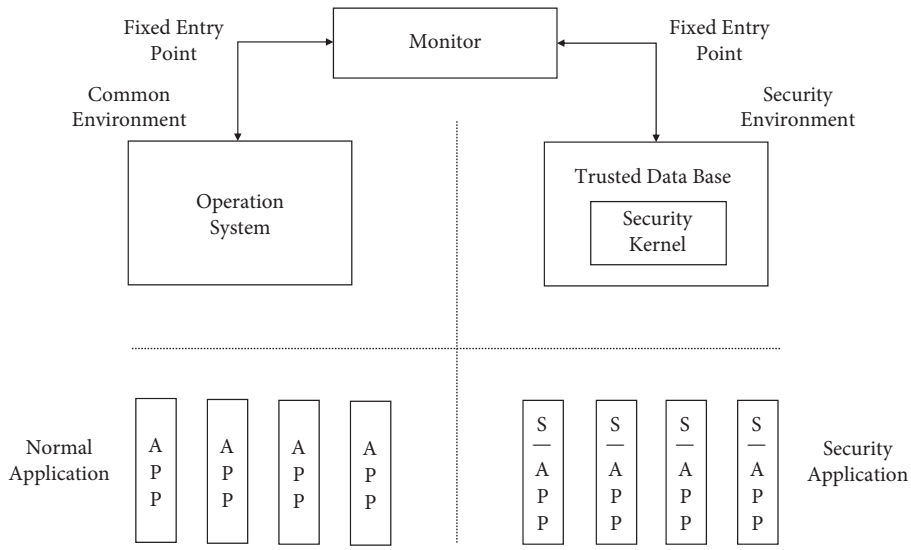


Figure 1: TrustZone technology system architecture [6].

running in REE, it cannot view or modify resources in TEE memory space, whereas a process running in TEE can view help in both areas [7].

To realize the physical division of memory, TrustZone designs two controllers: TrustZone Address Space Controller (TZASC) and TrustZone Memory Adapter [6]. TrustZone Address Space Controller can divide the memory address space of the device into multiple memory spaces. These intervals can be configured as the address space of TEE or REE through the security software running in TEE. TrustZone Storage Adapter is responsible for partitioning the static RAM or ROM of the device and extends some coprocessors. These coprocessors can extend the core functions by expanding the instruction set or providing configuration registers. Some coprocessing registers have one in TEE and one in REE. These registers only work in the corresponding environment and can be modified only in the corresponding environment. Other registers are global registers, but the restrictions on these registers are stringent. They can only be read and written in TEE, and only read-only permission is provided in REE [6, 8].

The last aspect of TrustZone hardware architecture is the extension of the interrupt control system. The interrupt is an essential part of TEE. It can prevent malicious software from attacking the system by entering the interrupt vector. In TrustZone, TEE and REE use interrupt input FIQ and IRQ as interrupt sources. If the interrupt occurs in the corresponding environment, switching the execution environment is unnecessary. For example, suppose the interrupt does not appear in the corresponding environment. If IRQ interrupts input is received in TEE, the monitor must switch the execution environment and close the interrupt. TrustZone uses a CP15 coprocessor [9] to ensure the safe interruption of internal resources. It contains a control register that the software can only access in TEE, thus preventing the software in REE from modifying the F bit and A bit in CPSR (F bit is used to mask FIQ interrupt, and A bit is used to mask external interrupt). In this way, CP15 can effectively prevent malware running in REE from shielding interrupts in TEE.

The hardware expansion of the above three aspects is the functional foundation of the TrustZone system architecture. We can see that TrustZone integrates security measures into SOC at design time and improves security without compromising previous chip designs. Similar hardware-based security technologies include XOM developed by Stanford University, and AEGIS developed by MIT. The core of the XOM system is its assumption that the internal units of the processor can defend against all kinds of malicious attacks. Namely, its TEE applies only to the CPU and not to the entire operating system. We can roughly think of XOM as the first aspect of TrustZone's hardware architecture for ease of understanding. AEGIS system is a security startup structure that provides multi-level security verification to prevent the invasion of malicious software during system startup.

2.2. Software Architecture. TrustZone hardware architecture extensions embed security into the processor, which provides the basis for separating security from the normal operating system (Rich OS, ROS) [8], i.e., a new secure operating system (Trusted OS, TOS) [10] can be implemented. In addition, the monitoring code area is added to realize the switch between TOS and ROS. TOS and ROS run on the same physical CPU simultaneously, and their interactions are limited to messaging and shared memory passing data [6]. TOS has independent exception handling, interrupt handling, scheduling, application, process, thread, driver, and memory management page tables [8]. The monitoring code area provides a virtual hypervisor that connects these two systems and stores and restores registers' states in both environments during the transition between these two systems. It ensures that the system can be re-executed during the transition to the new environment. To ensure the entire system's security, it must be guaranteed

from the start of the system boot. Many attackers attempt to erase or modify system mirrors stored in FLASH during system power outages. Because of this, TrustZone is started securely. The process is as follows: After the device is powered on and reset, a security boot program is run from the ROM of SOC. The boot program will first enter the initialization stage of TEE and start TOS. The critical codes in each stage of TOS startup are checked step by step to ensure the integrity of TOS and prevent the operation of unauthorized or maliciously tampered software [11]. Then, running the REE boot program and starting ROS are performed to complete the safe boot process for the entire system. ARM also defines the standard application program interface (TrustZone API-TZAPI). Such standard application program interface ensures that the applications written by software and hardware developers can be applied to devices on different security platforms and allows client applications to access TOS to achieve the purpose of managing and using security services.

2.3. Implementation of TrustZone Security Mechanism. TrustZone technology introduces the concept of TEE through the hardware expansion of the CPU core and memory system. The nonsecure (NS) bit is a crucial extension of the system, which indicates whether the current system is in a secure area or not. When the NS bit is set to 1, it means the current area is nonsecure and 0 means secure [6]. NS bit acts on the CPU core and memory system and affects the work of system peripherals. The monitor switches between the security state and the normal state by modifying the NS bit. Moreover, the monitor also extends the functions of cache and MMU (Memory Management Unit) in the memory system and adds their control logic to realize memory management. The monitor adds an NS bit to the tag of each cache line so that the data in the cache can be marked as secure and normal by the NS bit. The monitor creates two virtual MMUs for the two virtual processor cores and adds an NS bit to each page table. The tag of each Translation Lookaside Buffer (TLB) corresponding to the page table is also increased by one NS bit. So far, all memory (except the preset shared memory) has realized the security and normal marking through the NS bit. All NS bits are combined for dynamic verification to ensure that only authorized operations can access the resources marked as security by NS bits.

To ensure the stability of the boundary security between TEE and REE (i.e., ensuring the processes in REE cannot access any resources in TEE), TrustZone adds unique control signals to each read-write channel on AXI-BUS: Bus Write Transaction Control Signal (AWPORT) and Bus Read Transaction Control Signal (AIRPORT) [6]. Therefore, when the CPU requests to access the resources in memory, it should send the target memory address to AXI-BUS and send the AWPOER and ARPORT to the bus to indicate whether the access is a secure transaction or a nonsecure transaction [8].

AXI-BUS protocol will set these two signals to 0 or 1 to indicate the security transaction. Then, the address decoder of the system will identify these two signals and use these signals to generate different address mappings according to the security status of the CPU. There are two situations: When both AWPORT and ARPORT are 0, the CPU is in a safe state. At this time, the CPU can access all registers in the normal world and security world. When AWPORT and ARPORT are 1, the CPU is in the normal state. At this time, the CPU can only access registers in the normal environment. When it attempts to access a register in a secure environment, the address decoder will refuse access and generate an error message that "The peripheral does not exist with this address" [8]. At the same time, TrustZone uses the AXI-to-APB bridge to protect peripherals and ensure that peripherals are connected safely. Therefore, the peripherals in the normal world cannot access the secure world, which firmly separates the peripherals in the two environments. TrustZone ensures the security of TEE and REE boundaries in terms of data and peripherals. Store essential data resources in a secure environment and use these data in the secure processor core to ensure that these data can be protected from malicious software attacks that may occur in the normal world. At the same time, sensitive peripherals are isolated in the hardware to defend against malicious attacks through peripherals. For the peripherals, recent research [12] propose a lightweight framework named TEE-Watchdog, which establishes MPU protections for secure system peripherals in TrustZone. TEE-Watchdog can ensure prevention of unauthorized peripheral accesses and use a manifest file to log the application misbehavior running in the TEE. TEE-Watchdog introduces a compact CBOR-encoded manifest file template for device vendors or manufacturers to use for specifying access policies. It also enables efficient behavioral logging of misbehaving software. But TEE-Watchdog still has disadvantages; according to the researcher's benchmark test, there is a 1.4% delay in latency of peripheral access due to TEE-Watchdog protections.

TEE implemented by TrustZone technology enables security measures applicable to many layers of a complex embedded system. Normal operations will run entirely in ROS (Rich OS) [7] without TrustZone providing security assistance. To achieve security in ROS, TrustZone uses the following three steps to build TOS [10] to perform operations that need to be encrypted. First, TrustZone executes the boot program to complete the configuration of TOS, and only the modules that pass the security verification are allowed to be loaded. Second, during the operation of the system, the security code area provided by TrustZone technology will process the security requests of the normal code area, save the security requests in the shared memory before processing, and only the requests that pass the security detection will be processed. Finally, processes marked as safe can be executed in TOS away from ROS.

3. Build Trusted Execution Environment (TEE) Based on TrustZone

This part will cover the TEE built based on TrustZone in detail. Global Platform developed the TEE standard based on TrustZone security technology. As introduced in Section 2, TrustZone is a system security solution that combines

hardware and software but is mainly based on hardware. It isolates the system into two simultaneous operating environments (i.e., security and normal) and provides complimentary applications through software architecture. TEE developed by Global Platform can be used as an independent execution environment to reside in the security zone on the processor supporting TrustZone [6]. To ensure the safe storage and use of sensitive data in TEE. The structure of TEE and REE built by the Global Platform based on TrustZone is shown in Figure 2 [13].

We can see in Figure 2 that in this system architecture, TEE is an independent running environment running simultaneously with REE in the system. Its purpose is to provide security services to REE and ROS in REE. TOS in TEE is responsible for managing the software and hardware resources of TEE, and it also includes monitors responsible for switching TEE and REE. TEE ensures the safe operation environment of TA (Trusted Application) and protects TA's resource integrity and access rights. Each TA in TEE is independent and cannot access without TEE's authorization. TEE must pass dynamic multi-level verification during its startup and maintain its independence from ROS. TEE client API is the underlying communication interface for client applications running in ROS to access data and services. TEE functional API encapsulates TEE client API to access security services in a programming model familiar to developers, such as encrypted storage and trusted storage [13]. TEE Internal API provides TA's programming interface, mainly including APIs for key management, secure storage, and trusted UI. Trusted UI means that when critical information is displayed, or key information data (bank password and account password), hardware peripheral resources such as screen display and keyboard input are entirely controlled and accessed by TEE, and the software running in ROS does not have access at this time. A typical and similar example is that when using iPhone to input a password, the IOS system will force the user to use the input method provided by IOS to prevent the third-party input method from stealing the password when the user enters the password. Finally, REE Communication Agent provides a bridge between TA and CA (Client Application).

TEE built by TrustZone is generally based on the TEE architecture standard of Global Platform. Wang [14] from the University of Electronic Science and Technology of China built TEE for the Android system as ROS, as shown in Figure 3. From this TEE architecture, we can see that TOS running in a safe state manages the software and hardware resources in TEE. Moreover, TOS is started up before the whole system is started, and then the security attributes of the device are configured and set, and the secure storage space is allocated. REE Communication Agent is implemented in the form of the TrustZone driver, while the TEE Communication Agent is a daemon. Shared memory is allocated in the memory area of REE. Its function is to transfer data between TEE and REE. REE transfers command parameters to TEE and receives the data returned by TEE. The whole Android system runs in REE, so the Android system cannot directly access the sensitive data stored in TEE and the peripherals marked as security. This can protect the security of sensitive data of system users, preventing the whole Android system from being attacked by malware.

4. Research Works Based on TrustZone

Aiming at the rising security problem of embedded systems and applications, companies have successfully launched a security solution based on TrustZone technology. Researchers also pay much attention to this technology.

4.1. Construct Security Platform Based on TrustZone. TrustZone (TZ) technology gives a basic frame of security. Developers can utilize TZ technology to construct a new security platform to fulfil different security requirements, such as payment, fingerprint identification, and DRM. In 2013 MWC, Samsung company released a new mobile security platform KNOX based on the Android system and TrustZone technology. It utilized Android SE to execute enforce access control policies to isolate applications and data on the platform, thus satisfying the requirement of security guarantee [15]. The system frame is shown in Figure 4 [16].

Android SE security mechanism should guarantee the integrity of the system kernel, or it will lose efficacy. In the KNOX system frame, the TrustZone-based Integrity Measurement Architecture (TIMA) is responsible for this gap [6]. It utilized the TrustZone hardware frame to efficiently divide memory and CPU resources into security and normal worlds. TIMA runs in a safe area and cannot be forbidden. TIMA continuously monitored the integrity of the kernel in real time. When it detects the kernel's loss of innocence, it will announce the company IT by Mobile Device Management (MDM). In this situation, it can cooperate with the safe launch and Android SE to establish the first security defense of kernel and core helper process.

Also, AMD company in 2013 introduced a new AMD Secure Technology called Platform Security coprocessor (PSP). It utilized TrustZone technology to divide the CPU into two virtual spaces to create a safe space for processing sensitive data on PSP. Other tasks were processed in normal world space. This method guarantees the safety of the storage and process of sensitive data and trusted applications and can protect the integrity and confidentiality of critical resources [17]. The PSP design architecture is shown in Figure 5 [17].

In Jan 2014, AMD published the first ARM frame server processor, Opteron A1100. It was a complete SoC with an integrated functional design rather than a CPU. Moreover, it's also the first server processor supporting the TrustZone security module, which plays a vital role in increasing the security on the server.

Apple company designed a highly-utilized TrustZone security frame and further published a Secure Enclave module based on it, which significantly solved the problems of how to encrypt, store and protect the user's fingerprint biological information. It was also responsible for verifying

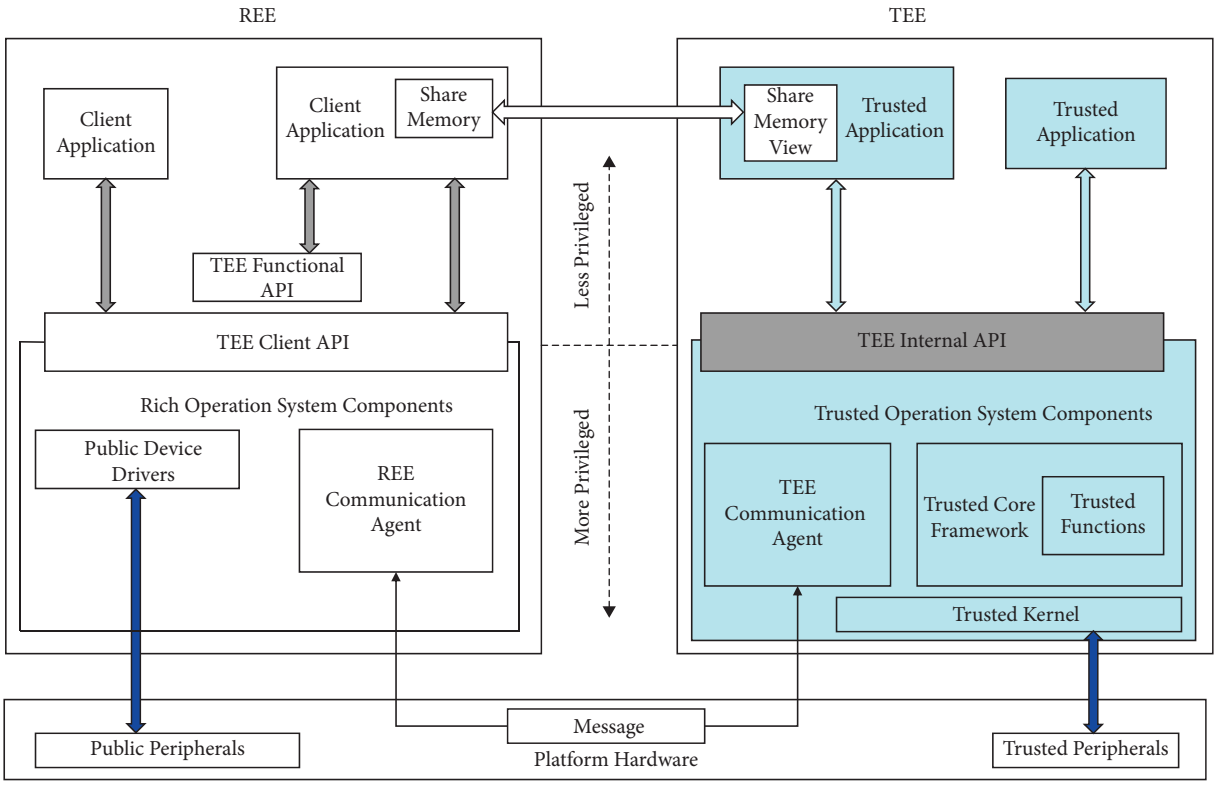


Figure 2: Global platform trusted execution environment system architecture [13].

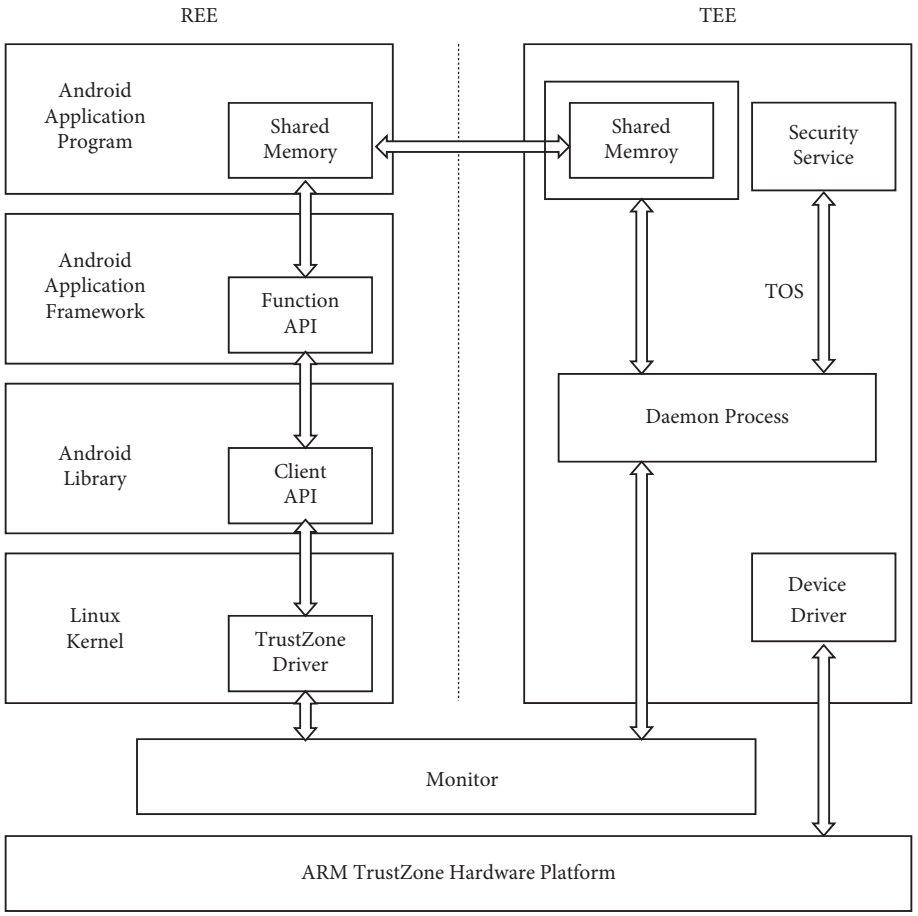


Figure 3: Trusted execution environment system architecture based on TrustZone [14].

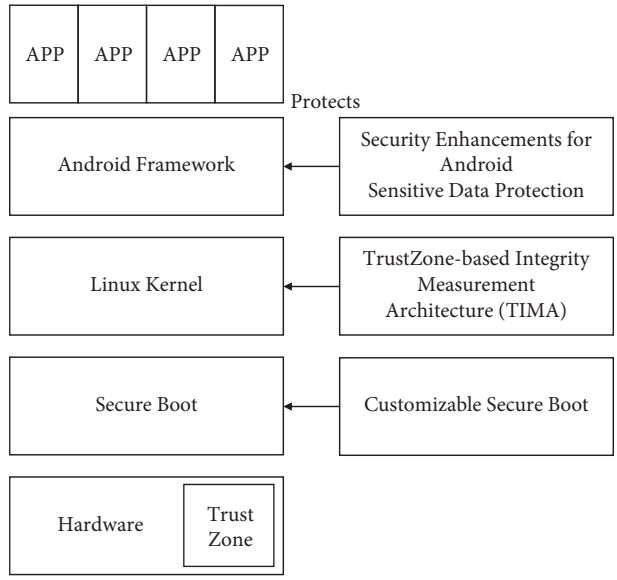


Figure 4: KNOX system security architecture [16].

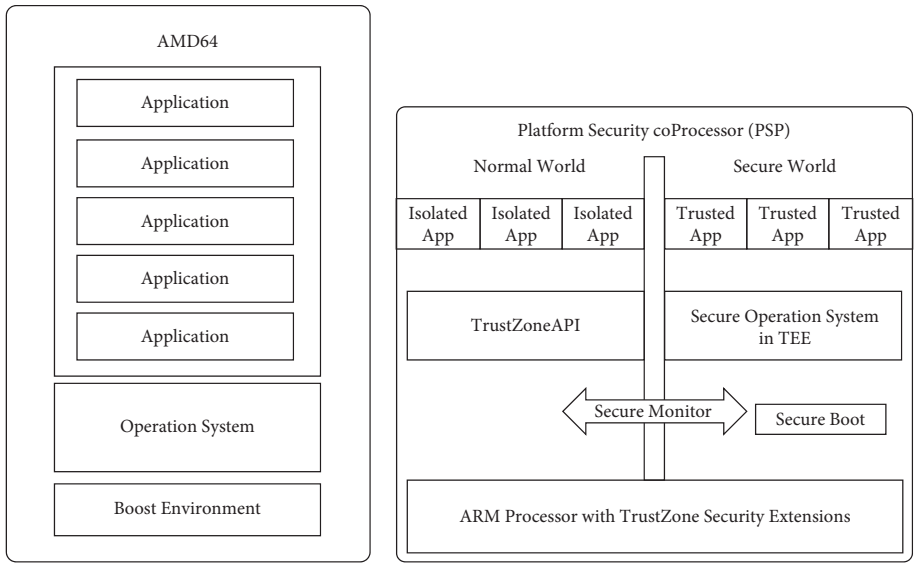


Figure 5: AMD PSP design architecture [17].

fingerprint information from Touch ID. The only match will permit visit or purchase. Secure Enclave module was the coprocessor internally built-in Apple A7 chip, with a security startup and software update mechanism independent of the central processor [15]. Companies' published security platforms and frames are based on a hardware security isolation environment supported by TrustZone to process sensitive data and security services. Their security is guaranteed because sensitive information processing, secure storage, and services are protected by TrustZone hardware and wholly isolated from ROS.

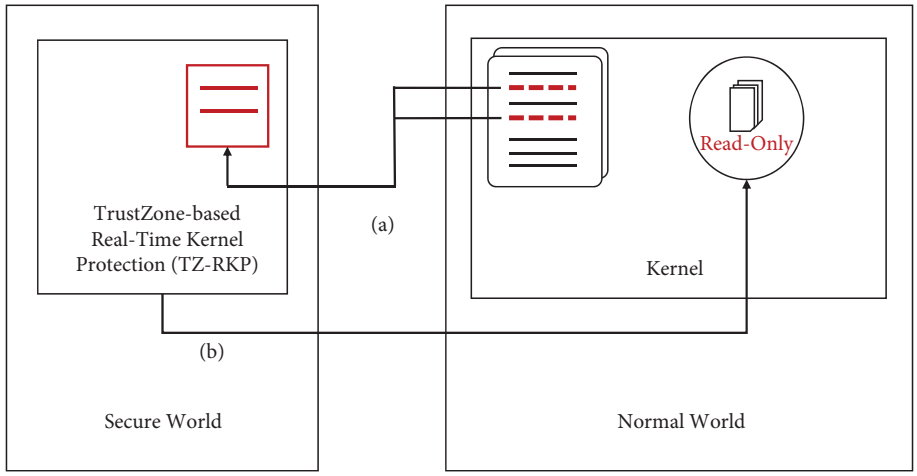
Nowadays, most security platforms in the industry are based on the hardware security isolation environment provided by TrustZone, which are widely being used to process and store sensitive information securely and design security services to meet application requirements. The TIMA in Samsung KNOX completes the integrity check of the Linux kernel, the PSP of AMD handles sensitive application information, and the Secure Enclave of Apple handles fingerprint biological information. They are all based on this idea, since sensitive information processing, secure storage, and security services are protected by TrustZone hardware isolation and completely isolated from ROS, its security can be adequately guaranteed. But KNOX uses TrustZone more as a monitoring system, constantly judging whether the kernel is intact, to confirm whether it is under attack. PSP and Secure Enclave module use TrustZone as the overall system architecture framework, which distinguishes the overall system environment into TEE and REE.

4.2. Construct Secure System Environment Based on TrustZone. With rapid development, system functions become more complicated. Code lines become larger. This increases many gaps, for which hackers can exploit them to gain sensitive data in the system. Utilizing the hardware security isolation advantage of TrustZone can guarantee the safety of the ROS system. Research [18] proposed a TrustZone-based real-time kernel protection (TZ-RKP) mechanism, such as the one shown in Figure 6 [18]. In the ROS kernel, control operations and page table update function will check in TOS rather than modify in ROS. It mainly coerces some privilege system functions in the kernel to review and authorize in a safe environment first, then permits processing. This mechanism can efficiently prevent the attack of modifying and adding kernel files. However, this paper did not implement the corresponding detection mechanism and processing mechanism in TZ-KRP for the attack that decoys the kernel to modify its data, and this kind of attack can hijack the kernel control flow and make it seriously damaged.

Research [19] focused on embedded systems and proposed a high-security system primitive platform, which combined TrustZone, TPM, and programmable security control logic (PSCL). This platform improved system performance and security. PSCL mainly consisted of three components: security finite state machine (SFSM), programmable security data path (PSD), and programmable security processing module (PSPM). TPM only interacted with secure kernel and provided secure storage, kernel integrity measurement, and system security integrity report. When the system processes trusted applications, it will deploy SFSM to check used and then define PSD to notify secure CPU based on SFSM. Secure CPU will choose suitable parameters and load them into the register. SFSM then constructs PSPM based on these parameters. When PSD and PSPM are created, SFSM will continually detect system status from the secure CPU. As long as the status turns unsafe, the secure CPU will reset SFSM and refresh PSD and PSPM. However, the security strategy, performance, and secure communication agreement between the security CPU and the SFSM were not anal yzed in depth. Research [20] constructed a security enhancement frame based on TrustZone and Linux systems. The structure consisted of an access control mechanism and a security enhancement method. The access control mechanism was implemented by Domain and Type Enforcement (DTE) module and improved Bell-La Padula (BLP) module. Furthermore, the security enhancement method employed the Linux Security Module (LSM) frame to provide vital protection for the system. The normal world used BLP and DTE policies provided by secure Linux to avoid malicious attacks and ensure the integrity and confidentiality of the system. At the same time, secure applications were processed by invoking security services within the TrustZone-isolated security environment. This prototype design can provide a secure execution environment for open embedded systems and various applications, but this paper did not combine specific application scenarios, and there was no specific implementation of security services.

4.3. Construct Trusted Computing Environment Based on TrustZone. TrustZone needs a trusted computing environment for its isolated environment to provide trusted computing functions for the system platform. Using this technique to construct MTM is a common way to build a trusted mobile platform, providing security for MTM [6]. The research work in [21] combined the trusted computing concept of TCG and a Linux-based embedded trusted computing platform. It built a virtual framework in the security zone of TrustZone, designed a trusted mobile platform prototype based on this, and implemented secure startup. This prototype implemented MTM in pure software aspect with no adding hardware and verified the feasibility of realizing embedded trusted computing software platform by utilizing hardware security mechanism. The work in [22] discussed two isolation environment construction software MTM by TrustZone, and other security components (such as JavaCard). With analysis, TrustZone can provide security protection similar to hardware MTM. The study in [23] put out a mobile trusted computing module TEEM, giving a trusted computing function for different platforms such as PCs or mobile devices. In this design, TEEM was settled as a TPM service running in the secure zone of TrustZone. This module didn't isolate TEEM and ROS, the same as TEEM running on the whole Linux OS, resulting in a vast TCB. From a trust computing perspective, the authors in [24] proposed and implemented a static measurement method of Android system trust based on TrustZone. In this method, the ARM Trusted Firmware (ATF) bl1.bin image is considered the root of trust, and the TrustZone technology is combined with the Android system knowledge base. Kernel modules and executable files are statically measured in the system boot process. Finally, the trust root is extended to the application framework layer of the Android system, providing a reliable underlying environment for detecting the application layer of the Android system. This method can detect the privileged attack promoted at the system layer of Android, and find the rootkit that undermines the integrity of the Android kernel in time during the boot process. In addition, the performance loss of this method is within an acceptable range.

The above technologies take advantage of the isolation environment of the TrustZone and build virtualization frameworks in secure world to establish trusted computing platforms. However, most of the above researches were not discussed in detail, and there were no effective testing and verification methods. TrustZone-based MTM, as mentioned in [22], did not have a valid MTM test suite to confirm that their implementation is fully compliant with the TCG specification. Meanwhile, TEEM mentioned in [25] was not isolated from ROS and had no effective verification results on the development board, so it cannot be combined with specific credible application scenarios. In addition, running TEEM on the entire Linux OS results in a large Trusted Computing Base (TCB). For method in research work [26], the researchers did not give different weight values to different files based on the boot relationship existing in the actual Android system. All these technologies need to be further studied and explored.



(a) Control instructions and page table update functions are replaced by traps to the secure world
(b) Page tables are mapped read-only so they cannot be directly modified by the kernel

Figure 6: TZ-RKP design diagram [18].

4.4. Construct Security Service Based on TrustZone. With the enriching requirement of embedded systems, new applications aiming at different needs occur. Applications that collect and process sensitive information from users have serious security problems. Based on the TrustZone security isolation environment, constructing a security service to protect sensitive data from malicious attacks is feasible. The research work in [23] utilized TrustZone technology to create a new mobile online ticket purchase system. Payment involves various sensitive information. In this system, sensitive information will be stored in secure zone of TrustZone hardware, which greatly guarantees the security problem when purchasing. The work in [24] aimed at low protection of privacy in payment system, proposed TrustZone-based privacy protection platform. It is suited in the online payment (such as NFC), with applications needing privacy protection. During online payment, applications can communicate with trusted applications through the internal TrustZone API mechanism and store sensitive data such as privacy during payment in a secure environment isolated by TrustZone. Meanwhile, security monitors will monitor these data to prevent privacy leakage caused by malicious attacks. However, the prototype design proposed in this paper was not implemented on the specific development environment, and the performance load will be very large which need to be discussed. The research work in [27] is focused on remote attestation, a mobile remote authentication solution based on ARM TrustZone (TZ-MRAS) is proposed, which builds a probe-based model for dynamic monitoring of system kernel and program integrity (ProbeIMA), which dynamically detects unknown fingerprints generated during kernel and process execution. Thus, it prevents attackers from using time-of-check-to-time-of-use (TOC-TOU) defect to create attacks. Compared with existing solutions, this solution is more secure, efficient and versatile, and more suitable for low-cost heterogeneous systems.

4.5. Construct Secure Startup Based on TrustZone. To achieve system safety, the secure startup is the base. This way, it can ensure that the system's operating environment is genuinely trustworthy. Because of this, there are a lot of studies on secure startup based on TrustZone. The authors in [28] reconstructed trusted root for the TrustZone platform to ensure startup security based on SRAM PUF [9]. The concrete implementation was as follows. Firstly, the building block is implemented in SRAM on-chip, which is mainly responsible for extracting the original seed (PS) and random number seed (TRS) from the initial response of SRAM. Where, PS is used to generate a unique device key, TRS is used to build a secure random number generator (RNG) for TOS. They are all based on the construction of fast roots. The building block also provides secure startup of TOS and security services. After that, the device key is used to provide encryption/decryption primitives for security services in TEE, and the TPM service of pure software is integrated into the TEE environment to provide rich TPM services for ROS, which effectively protects ROS from software attacks as the secure root of ROS. Thus, a trust chain is formed from system startup to normal operation, and ROS applications can use TPM service to extend the trust chain to the application layer. At the same time, the overall design effectively protects the system's security. This research was based on the memory isolation mechanism provided by TrustZone, which completely isolated the trusted root from ROS and avoided software attacks from ROS. However, its main memory was not implemented outside the SoC, so the designed trust root cannot resist physical attacks directly attacking the hardware platform.

With growing wireless sensor network applications, such as intelligent home systems, transportation systems, and long-distance military systems, security issues have become increasingly prominent. The most important is the physical attack during the startup stage of the device. Based on this, research work in [29] analyzed that most working wireless

sensor nodes rely on software to protect system security. But with its development in military and medical aspects, security protection based on software seems lacking and insufficient. Therefore, this research is based on the features of hardware isolation and secure memory configuration provided by TrustZone and proposed a new secure startup system for wireless sensor nodes. However, the research did not implement and analyze security specifically. Another research in [30] also analyzed secure and trusted startups' existing problems. The secure startup was designed for particular devices, and users cannot choose software. Moreover, trusted startup lacked a check mechanism when processing. Based on TrustZone technology, which can support security hardware isolation, this research proposed twice start validation architecture, which significantly solved the above two weaknesses. Specifically: phase 1 startup verifies the boot program and OS image and registers them; phase 2 running applications verify startup traces and if the running software meets security conditions.

4.6. Construct Virtualization Platform Based on TrustZone. System virtualization (VMM/hypervisor) provides an excellent isolation processing environment for applications with many problems. From a generalized aspect, TrustZone is also a kind of virtualization technique. Compared with VMM, TrustZone provides hardware isolation and a memory protection mechanism [6]. Thus, utilizing TrustZone can enhance the security of existing software virtualization techniques, overcoming the weakness. The work in [31] focused on the percent virtualization technique that simulates critical instruction leading to high load. Based on TrustZone's privilege mode and user mode in the normal zone and secure zone, ViMoExpress was proposed. ViMoExpress is a lightweight virtualization solution for embedded systems. It will run in Monitor mode, which significantly decreases the load. This design highly increases the efficacy of a single-core ARM processor with few code lines. The load is about only two break times of system switch. Research in [32] is based on TrustZone security, an expanding implemented asymmetric virtualization layer that supports a single-core processor simultaneously runs RTOS and GPOS. This implementation does not need to modify GPOS or use privilege instruction to decrease processing load. At the same time, another research in [33] is based on TrustZone virtualization, where authors proposed a new software framework SafeG as a monitor without modifying the normal operating system (GPOS), as also shown in Figure 7 [33]. This framework implemented GPOS and RTOS processing simultaneously on a single processor. Inside RTOS will process tasks with the high real-time requirement, while GPOS process user normal tasks. SafeG isolates external devices to allocate them to the required operating system in real-time, which reduces the isolation load of RTOS and increases its operational reliability. This design is incredibly suitable for scenarios such as car navigation systems, mobile phones, and machine tools from these advantages.

4.7. Brief Conclusion. The commercial applications and academic research on TrustZone technology have been involved in all aspects of embedded systems, providing a reliable guarantee for the system security of embedded systems. Based on research and development works, whether the commercial application or academic research is critical to solve the security problems existing in the embedded system. Based on the system-level security framework provided by this technology, we can develop a security platform to meet specific needs and design a new security policy.

5. Analysis of Advantages and Disadvantages of TrustZone Security Technology

This part compares TrustZone with several other technologies related to improving system security, focuses on the advantages and disadvantages of TrustZone, and introduces the optimization scheme in the next part.

5.1. TrustZone Security Architecture Analysis. TrustZone technology constructs a security isolated operating environment for the system, isolating the potential security threats of untrusted software, running the isolated software normally, and monitoring its behavior. It solves the security threat of the system against untrusted software. At present, there are three main methods to establish security measures in mainstream embedded systems: SOC external hardware security module, SOC internal hardware security module, and software virtualization technology. The advantages and disadvantages of the three solutions are analyzed in detail and compared with TrustZone.

The first method is the external hardware security module. Its core is to add a particular security hardware module externally in the system design, such as the SIM card in the mobile phone or the smart card with access conditions in the TV set-top box. This method can protect sensitive data in a solid physical device. Due to the completely independent design and production process of separate modules, more advanced tamper-proof and physical security technologies can be fully considered in the design and production process. However, this method increases the design cost of SOC, increases the system's power consumption, and has the probability of reducing the comprehensive performance of the processor. Another disadvantage of this method is also apparent. It only provides the functions of secure processing and secure storage. Still, when the software runs outside of the secure hardware module to process sensitive data, it is easy to make the data attacked by malware.

The second method is to build a hardware security module in the system, which mainly has two forms: one is to add a hardware module to manage encryption operation and secret key storage, and the other is to build a hardware security module on the general processing engine in the central processor. The former prevents unauthorized applications from accessing sensitive resources using internal security hardware logic. Compared with the first method, this method sacrifices hardware security but reduces the chip

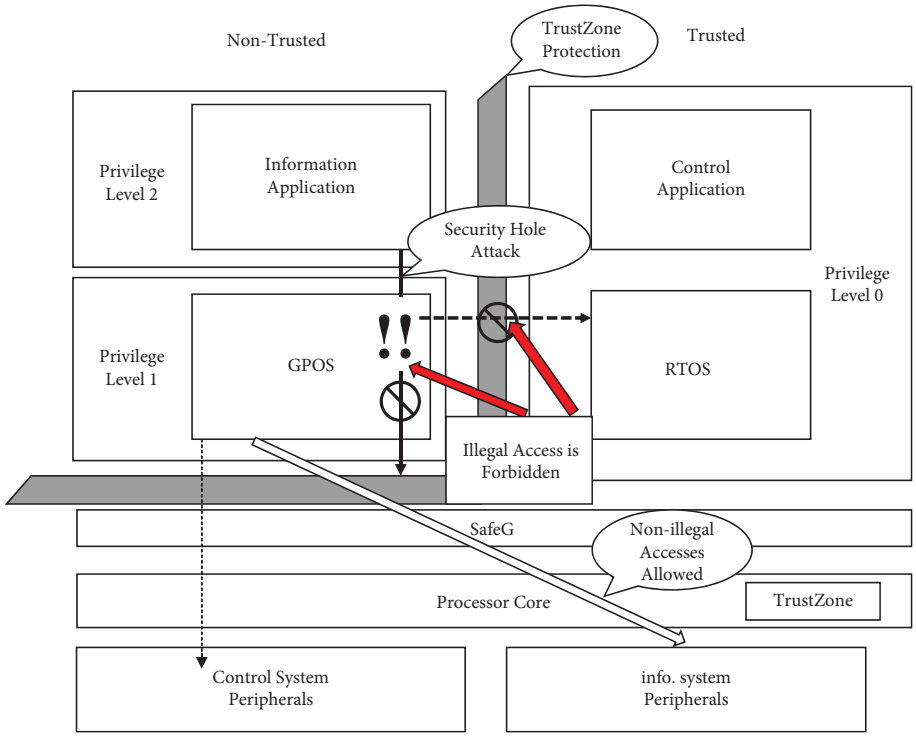


Figure 7: VMM design framework based on TrustZone [33].

design cost and facilitates integration. The latter provides a particular general-purpose processor for the security subsystem. This method is similar to the hardware security solution in TrustZone, but it also has shortcomings. The first point is that the design requires a separate physical security processor, which will increase the power consumption of the system and the area of the chip. At the same time, because the communication between the security processor and the general processor needs to refresh the data in the shared memory frequently, and the shared memory is usually external for security reasons, it needs to occupy a lot of execution time. The method of internal hardware safety module can only ensure the safety of system functions but does not consider the system safety of SOC in debugging mode and test mode. The system is particularly vulnerable at this time, but turning off the debugging mode and test mode will inevitably make it very difficult to diagnose software and system problems [8].

The third approach is software virtualization, which provides an isolated execution environment. The working principle of software virtualization is based on special software called VMM. Multiple virtual machines managed by VMM run independently in an isolated environment and will not be disturbed and destroyed by other virtual machines. Therefore, security-sensitive software can be moved to run in a secure environment running in VMM, while normal software can run in a nonsecure environment. However, VMM technology ignores the attacks related to hardware attacks, such as those suffered in debugging mode and test mode. To ensure the security of the virtual system, debugging must be disabled, and the test must be completely invisible, which makes it very difficult to develop software and diagnose software defects. In addition, some bus masters, such as DMA engines [34] and GPU, can bypass the protection mechanism provided by VMM. At the same time, virtualization technology also faces many vulnerabilities because it needs to do a lot of work in system management and resource allocation. It also increases the system execution load because it needs to simulate critical instructions.

5.2. Analysis of Advantages and Weakness of TrustZone. In the above comparison, we find that TrustZone has apparent advantages over these three security solutions. We can summarize the advantages of TrustZone. TrustZone technology is a highly secure system architecture designed through a reasonable combination of hardware and software, which hardly affects the system's actual power consumption and performance. Therefore, this technology has many technical and commercial advantages in improving embedded system security, mainly divided into the following aspects. First, it can provide a secure isolation environment for on-chip confidential data, and this processing method is also the best way for confidentiality. For example, suppose you want to use a CPU on the SOC to process the key in the SIM card. In that case, you must ensure a completely secure area in the SOC environment, and an operating system with low security cannot complete this operation [3]. Secondly, performance has always been an insurmountable problem in some security systems, especially the frequent transmission of encrypted information between the on-chip processor and off-chip memory. At this time, TrustZone can play a role because it can ensure complete bus bandwidth for the whole storage space, while the data in its security buffer can be

stored in clear text to achieve fast access. The encrypted data can be ordinarily stored in flash memory so that some cheap and flexible storage methods with large capacity can be used. In addition, TrustZone system architecture is a reasonable combination of software and hardware. Even after the SOC design is completed, it can still ensure that users can flexibly customize and upgrade the security system. Finally, TrustZone defines a secure isolation environment in the embedded system, containing some direct peripheral channels, such as user interface, Sim card, Smart card, and audio output. TrustZone provides security for all aspects of SOC devices through integrity checking mechanisms for unsecured parts. For example, decoded DRM audio data transmitted to an unsecured area can be protected by integrity detection of relevant components of the operating system [3].

However, TrustZone is not omnipotent, and its shortcomings are also undeniable, mainly reflected in the following aspects. First, it can only defend against various software attacks, but it is challenging to prevent physical attacks, such as physical tampering with the device's main memory. In addition, while it can ensure the security of isolated kernel code through a metric mechanism and periodically checks the integrity of the ROS kernel, there is no absolute protection against malicious attacks on the system at this point because attacks have already occurred. Secondly, it only provides an isolated execution environment without proving the credibility of the environment to users or remote users. Finally, providing a reliable, trusted root for the system platform is the cornerstone of the whole system's security. At present, this technology is based on solidifying the device key as the root in the system on chip, which will be difficult to update the key. Once the critical leaks, the whole platform will become unusable. TrustZone needs to store the device key on the device for a long time, so its security is difficult to guarantee, such as how to defend against bypass attacks, fault attacks, reverse engineering, and other types of attacks.

Therefore, the following key consideration should be how to provide a trusted root that can defend against both physical attacks and software attacks without adding hardware to TrustZone's existing hardware security infrastructure to ensure that the system runs in a trusted execution environment from device startup to operation. Of course, facing the increasingly stringent security requirements in the embedded field, it is also essential to ensure that the execution environment based on TrustZone security isolation not only provides various sensitive data processing and security services but also ensures that its TCB is as tiny as possible, to ensure the security of TOS. Although Trusted Computing Group (TCG) combined hardware and software to achieve a more secure computing environment, it released Trusted Platform Module (TPM) suitable for PC platforms [35]. Both Intel and AMD specify TPM as the trusted root of Late Launch, but the TPM hardware module is not eligible for embedded fields with strict chip area and power consumption requirements. Subsequently, TCG released the Mobile Trusted Module (MTM) for the embedded system and introduced trusted startup. Unfortunately, the function of MTM is not hardware implementation [24]. Some literature studies have also reassessed the performance of MTM and found that MTM will increase the power consumption of the system and reduce the performance of the encryption function. Therefore, taking MTM as the trusted root of embedded systems is not ideal. However, SRAM PUFs (Physical Unclonable Functions) could be more effective to extract the key as trusted root. It directly integrates the physical characteristics of the device SRAM, so no additional hardware resources are required, and the TCB of the device can be reduced. In addition, as a critical extraction and secure storage, SRAM PUF technology can resist various hardware and software attacks, such as reverse engineering and effective cloning prevention, to solve the problem that TrustZone is difficult to defend against physical attacks to some extent. In addition, it can also be used to construct a random number generator without the influence of hardware construction cost, performance, and power consumption. However, TEE isolated by TrustZone technology does not provide users or remote users with evidence that the software running in TEE does not tamper with malicious codes. Combined TrustZone can realize this technology with the small proof function provided by MTM software. TrustZone ensures that the software MTM runs in a secure, isolated environment, and MTM remotely proves that TOS is trusted. In addition, it can also provide rich delegated computing functions for the system platform, such as secure storage, identity authentication, and platform security protection. Of course, while considering the system security, it is necessary to fully assess the application requirements and design for specific application scenarios to ensure that the TCB of TOS is small enough to realize the system security in the real sense.

6. Security Performance Optimization

This section discusses possible ways to optimize the security performance of TrustZone technology. In order to find the direction of optimization more precisely, this section first analyses the formula for the total delay of the program calling TA in the TrustZone technique. After that, find the factors that affect the performance and propose the optimization methods through the formula analysis. Finally, this section explores the state of the art and analyses the feasibility and effectiveness of the mentioned performance optimization methods.

6.1. Problem Analysis. The TrustZone-based application structure has been changed compared with the previous single-core system. The code has been divided, and the two worlds have been added. The process of calling TA from CA is shown in Figure 8 [5]. In the program execution process, mathematical modelling is carried out for the TA process of the CA call. Considering that the program execution needs to go through several TA calls, the number of calls is related to the size of shared memory allocated by registration; Mg is the size of shared memory allocated by registration; M is the size of data transferred. N is the number of times a CA

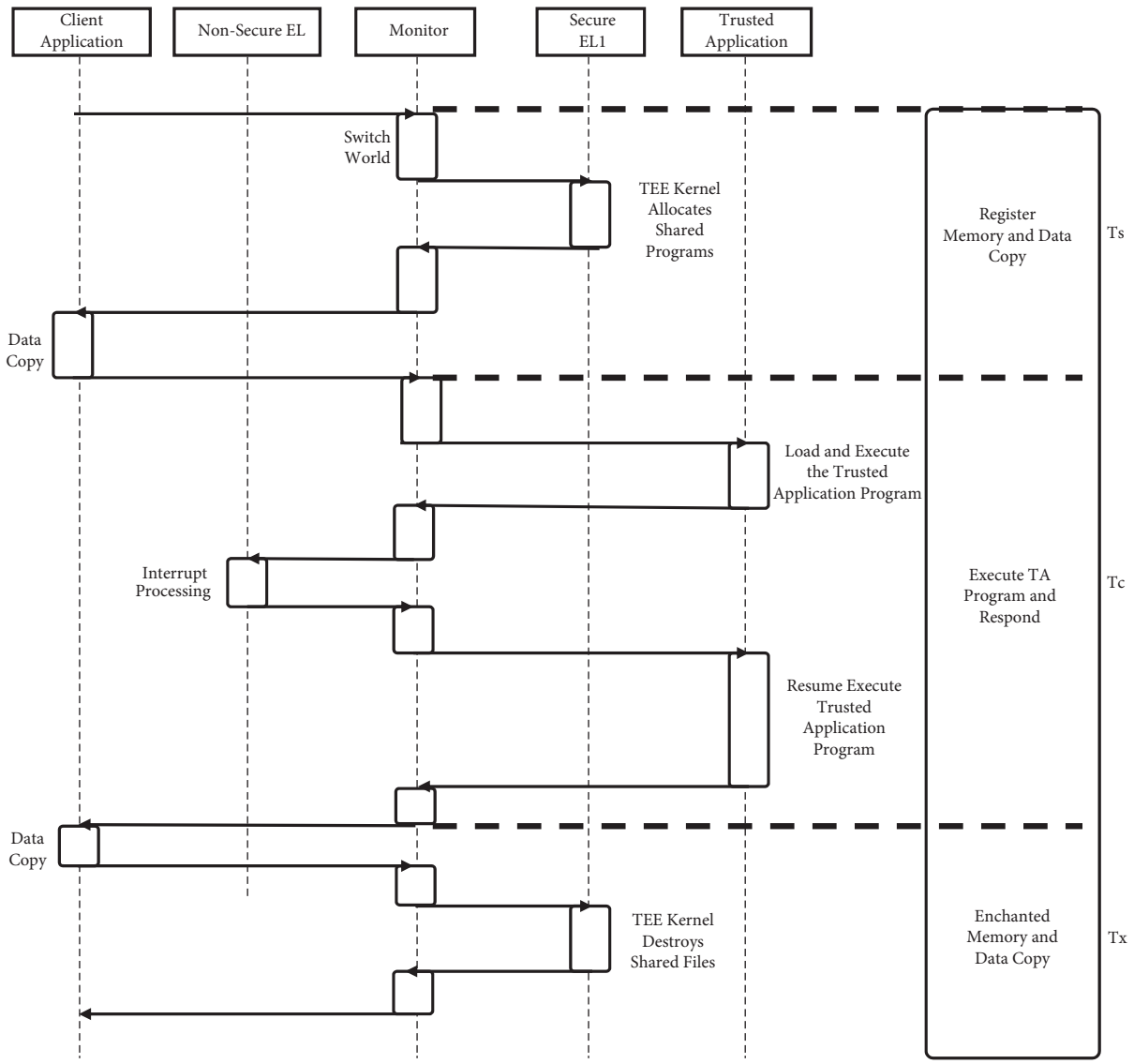


Figure 8: Service request process from CA to TA [5].

executes TA during program execution. These three satisfy the following [5]:

$$N = \frac{M}{Mg}. \tag{1}$$

To decompose the specific call process, it is necessary to register shared memory and data copy for input and output data, respectively, before calling the command, which will undergo world switching and data copy. T_q is the execution world switching time, T_c is the data copy time. If the execution time of the two shared memory registration and data copy processes is T_{s1} and T_{s2}, respectively, then [5]

$$\begin{aligned} T_{s1} &= T_{q1} + T_{c1}, \\ T_{s2} &= T_{q2} + T_{c2}. \end{aligned} \tag{2}$$

After registering the shared memory and completing the data copy, the CA calls the TA program, which will go through world switching, performing external interrupts, and executing the TA program. Assume it costs T_c when CA calls the TA program. The break time in the outer part of TA is denoted as T_i, and the specific execution time of TA is indicated as T_a, then [5]

$$T_c = T_{q3} + T_i + T_a. \tag{3}$$

After CA calls and executes the TA program, data copy, and shared memory destruction are required. T_{x1} and T_{x2} are the execution time of the first and second shared memory destruction, respectively. They satisfy [5]

$$\begin{aligned} T_{x1} &= T_{q4} + T_{c3}, \\ T_{x2} &= T_{q5} + T_{c4}. \end{aligned} \tag{4}$$

Therefore, the total delay T of program calling TA satisfies [5]

$$T = n * \left(T_{s1} + T_{s2} + T_c + T_{x1} + T_{x2}\right). \quad (5)$$

6.2. Optimization Methods. From the above analysis, to guarantee the security of the program execution process, compared with the program in normal system, the application based on TrustZone technology adds additional procedures, e.g., registering and destroying shared memory, data copying and interrupt execution and world switching. Consider the formula analyzed, to improve the performance of the call, it is necessary to ensure the execution time of TA while reducing T_{s1}, T_{s2}, T_{x1}, T_{x2}, T_q, and T_i, which are non-TA execution times, and also reducing the execution times N of calling TA and reducing the world switching times.

Therefore, researchers in [5] proposed the following methods to optimize and improve TrustZone security application performance: (1) reasonably configure parameters to reduce unnecessary calls and redundant execution as much as possible; (2) configure flexible interrupt processing modes, fully consider the requirements of different application execution processes, respectively provide shielding and unshielding interrupt request interrupt processing modes in TA execution process; (3) optimize the shared memory allocation mode, providing "one-use" and "one-multi-use" memory allocation mode; and (4) reduce the internal storage copy shell, use pointer to transfer data.

6.2.1. Reasonably Configure Parameters to Reduce Unnecessary Procedure Calls. Improper parameter setting will introduce unnecessary state switching or redundant procedure calls and execution, which will affect program performance. During parameter configuration, minimize the number of calls. From formula (1), the calling times are mainly reduced from the following three aspects: (a) Increase the value of Mg. When transferring the same data, the requested shared memory increases and the data transfer capacity increases, thus reducing the number of calls. (b) The applications shall request shared memory equal to the amount of encrypted data or an integer multiple of the required data size. That is

$$\begin{aligned} M &= n \times \mathrm{Mg}, \\ M &= n * \mathrm{Mg}. \end{aligned} \quad (6)$$

This reduces the waste of shared memory space on the one hand and reduces the time to execute redundant processes on the other hand. (c) Reduce the number of calls in other links as much as possible. Consider registering and destroying shared memory only once. Thus, in formula (5), the T_c between CA invocation and TA is unchanged, the shared memory time T_{s1} and T_{s2} for two times of registering become the shared memory time T_s for one time, and the shared memory time T_{x1} and T_{x2} for two times of destroying become the shared memory time T_x for one time, which reduces the number of invocation execution and improves program execution efficiency. New formula:

$$T = n \times \left(T_s + T_c + T_x\right). \quad (7)$$

6.2.2. Configure Flexible Interrupt Handling Modes. According to problem analysis, responding to interrupt requests will reduce system performance. When performing TA, blocking interrupts also face the situation where other applications in the REE wait too long and starve to death. This article takes a flexible approach to address different application requirements. The two interrupt response execution modes are Execution 1: CPU shields interrupt requests from REE before entering the TEE system and loading security application TA thread. After implementing the security application program TA, open the middle fault request from the REE system. For the interrupt request from the REE system, the CPU suspends the execution of the safe application TA and saves the thread state and TEE system state of the secure application TA. Then, switch to the REE system to perform the interrupt request. After the execution, the system state of TEE is restored, the saved thread of safe application TA is loaded, and the safe application sequence TA continues to run. After all security application programs, TA is executed, switched to REE system and executed common application program CA in user mode. In this way, for TA programs that are implemented for a long time and have low priority, the CPU requirements of REE end applications in the execution process are considered, and the situation of "starvation" of REE programs is avoided. Execution 2: the CPU shields interrupt requests from REE during the entire process of calling the secure application TA. Until the completion of TA execution, switch to the REE system and return to the execution of common application CA in user mode. The normal application CA reads the execution results produced by the security application TA from the shared memory. The response interrupt time is removed in the TA execution process, and the program execution efficiency is improved. This execution mode is suitable for the TA program with short execution and high priority, as shown in the following formula:

$$T_c = T_{q3} + T_a. \quad (8)$$

6.2.3. Optimize the Shared Memory Allocation Mode. Through problem analysis, the application program using TA frequently has a lot of repetitive system world switching and data copy problems. However, the system does not want the TA application program to occupy the shared memory resources for a long time. Therefore, two shared memory allocation and reclamation methods can be optimized: One with One-Use and One with Multi-Use. In One with One-Use mode, the shared memory is not allocated in advance when the CA calls the TA. The system registers and allocates the shared memory based on program requirements. After the secure application TA is executed, the system automatically copies the result data from the shared memory to the common application CA data area and destroys the shared memory automatically. In this way, the CA code of

common application programs is reduced. Users do not need to pay too much attention to the allocation and reclamation of shared memory. The situation of occupying shared memory resources for a long time is avoided. This works well for cases where the secure application TA does not need to be called multiple times. In One with Multi-Use mode, the user registers the shared memory according to the requirements of the secure application TA before invoking the shared memory. After the secure application TA is executed, the user decides whether to destroy the shared memory. In this way, it reduces the situation of repeated registration allocation and destruction of shared memory in calling multiple secure application TA. Moreover, it also reduces the case of multiple copies of data in shared memory, especially suitable for a general application program CA, which needs to repeatedly call secure application TA and deal with large data volume.

6.2.4. Reduce Memory Copy. Through problem analysis, when data is transferred to each other through shared memory, it faces the situation of multiple data copies. Improve program performance by changing to use pointer through the data-calling procedure. Essentially, after a CA registers and allocates shared memory, the original data is read directly into the shared memory. Only Pointers to data blocks in shared memory are passed when a call is made to the executing TA pass parameter number. When the encryption process is performed in TEE, the data is read directly from the shared memory. After the encryption, the resulting information is also put directly into the shared memory. This reduces the number of in-memory data copies and improves program execution efficiency.

6.3. Experimental Results. In [5], the authors conducted the experimental verification in the following environment: Hikey 960 development board, 4-core CortexA73 plus 4-core CortexA53 processor, 4 GB RAM, Linux kernel version 4.13.0, OP-TEE version 3.3.0, 2 MB of REE and TEE shared memory. The test procedure was recorded and organized by using Ubuntu 16.04. Research [5] implemented the AES encryption procedure in OP-TEE, and tested the correctness of the problem analysis and the effectiveness of the improvement method by testing the number of CPU cycles consumed by the CA in the AES encryption procedure when calling the TA to perform encryption. The validity of the problem analysis and improvement method is verified by testing the number of CPU cycles consumed by the CA in calling the TA to perform encryption in the AES encryption program. The CPU cycles consumed during the execution of AES calls are also tested in detail with the help of a CPU performance monitoring unit (PerformanceMonitorUnit).

By analyzing the experimental results reported in [5], we can state that: (a) System world switching affects program execution efficiency. If the parameters are not set properly, the CPU cycles consumed in the world switchover phase account for 49% of the entire execution process, which seriously affects the execution efficiency of the program. By adequately setting parameters, reducing world switching, eliminating procedure calls and redundant execution, you can achieve up to a 31% improvement in performance. (b) Responding to external interrupt requests will cause uncertainty in program execution and reduce the conductance energy by 60%. The maximum performance of interrupt requests can be increased by 4.5%. (c) Register and destroy shared memory includes memory allocation and reclamation, as well as the copy of memory data. In the case of no memory overcommitment, the number of CPU cycles consumed in the phase of registration and destruction of shared memory accounted for 27% of the entire execution process, which reduced the sequential performance of the program. Memory overcommitment can achieve a maximum performance improvement of 37% for extensive computation services. (d) Reducing in-memory data copies can achieve a maximum performance improvement of 39%.

7. Analysis between ARM TrustZone and SGX

ARM TrustZone and Intel SGX are mainstream TEE technologies that aim to create a secure and isolated environment for sensitive task computing and private data storage to prevent attackers from obtaining data and harming system security. They are used on different platforms and application scenarios, so they adopt other design concepts, making a massive difference. This section analyses the security protection of ARM TrustZone and SGX from the perspectives of their design concepts, isolation protection principles and operation mechanism.

7.1. Design Concept. ARM TrustZone and SGX both guarantee a trusted execution environment at runtime, so that malicious code cannot access and tamper with the protected content of other programs at runtime, enhancing the security of the system, but they adopt different design concepts, as shown in Figure 9 [36]. The design concept of TrustZone is based on the CS model design, which constructs two separate worlds, set the Trusted Execution environment for the secure world. The SGX is based on the P2P model design, as shown in Figure 10 [36]. The CPU in TrustZone works in the secure world and the normal world, and the two worlds communicate with each other through SMC instructions. But in SGX, a CPU can run multiple secure enclaves and can run parallelly, and the memory occupied by the enclave will encrypt the hardware. These two different design models cause a massive difference in the design concept between TrustZone and SGX, which is reflected in Trusted basis design, Security service design and Task scheduling [36].

7.1.1. Trusted Basis Design. The TrustZone uses the entire secure world as the trust base, including security components, secure operating systems, and secure applications. Normal world applications share the same trust base. The secure world is configured by device manufacturers, thereby simplifying user development and use. However, there is a lack of effective isolation between secure applications in the secure world. The failure of any secure application will lead

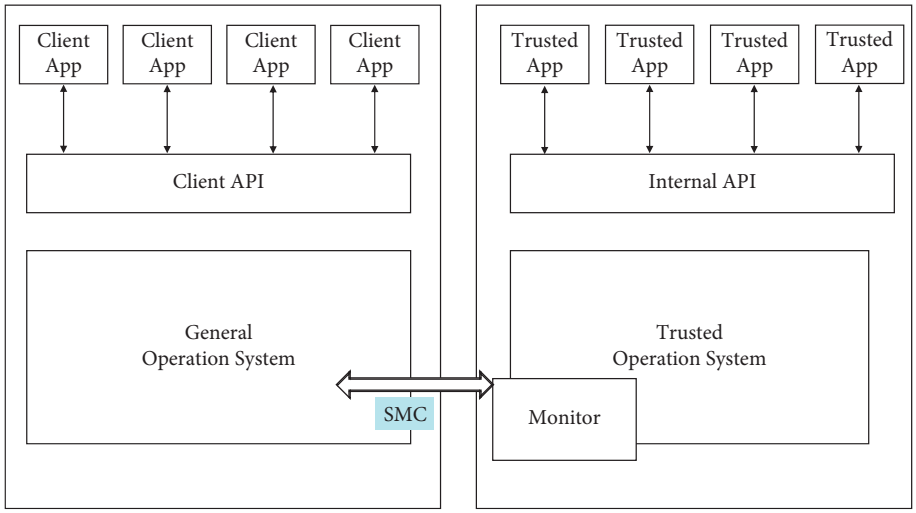


Figure 9: ARM TrustZone schematic design diagram [36].

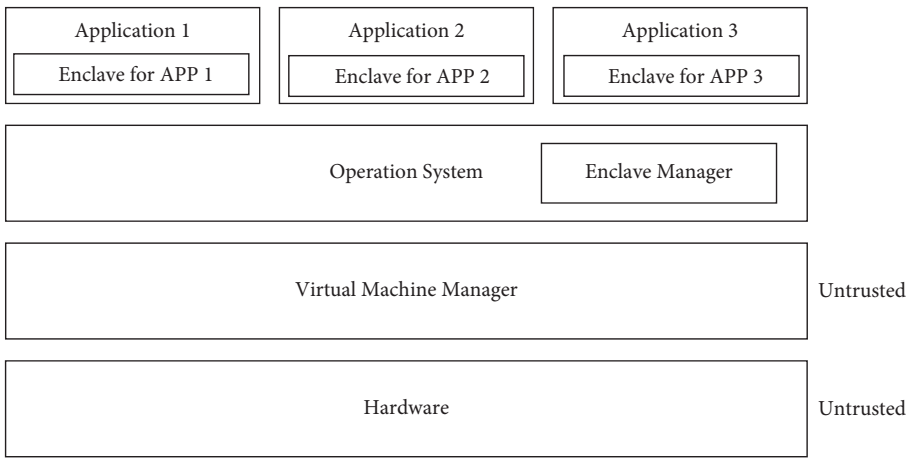


Figure 10: SGX schematic design diagram [36].

to the loss of the entire trust base. SGX regards enclave as an independent, trusted base, which corresponds to applications one by one. Enclave does not pose a threat to the system and other enclave security but increases the difficulty of user development and maintenance.

7.1.2. Security Service Design. TrustZone deploys vendor-designed secure code in the secure world ahead of time, and normal world applications can only request fixed generic security services through vendor-provided security interfaces. TrustZone can not provide dedicated services for the normal world beyond secure code. SGX security service is more specialized. SGX applications are divided into secure part and nonsecure part, which are developed by users. The secure part is deployed in the enclave and runs in isolation, creating different secure codes according to different functions.

7.1.3. Task Scheduling. TrustZone and SGX are designed for multi-core systems and support virtual machines. The TrustZone processor makes only one secure call at a time per core, supported by the Monitor module for world state switching. SGX endorses the execution of multiple enclave threads, and the running process processor can respond to interrupt execution, so the task scheduling is more flexible than TrustZone.

7.2. Isolation Protection Principle. ARM TrustZone and SGX adopt different Isolation protection principles. TrustZone provides a secure world that operates independently of the host which contains all secure operations. Since TrustZone is only divided into secure world domain and normal world domain, TrustZone only needs to formulate isolation protection policies around the secure world, which uses software and hardware to divide resources between the secure world and the normal world. In software, a dedicated operating system in the secure world is a complex, but powerful, design. It can simulate concurrent execution of multiple independent secure world applications, runtime download of new security applications, and secure world tasks that are completely independent of the normal world

environment. Secure bits are extended on hardware to isolate resources such as memory and I/O using auxiliary controllers such as Adavanced eXtensible InterfaceAXI, TrustZone Address Space ControllerTZASC, TrustZone Memory AdapterTZMA, and TrustZone Protection Controller to provide hardware resources required for the operation of the secure world. The Monitor is a security critical component, as it provides the interface between the two worlds. The Monitor module runs in the highest privileged state. The Monitor module runs in the highest privileged state. It connects the normal world and the secure world, isolates access between different worlds, and provides system-level secure protection.

SGX allows users to actively create and maintain enclaves, deploy and apply secure code and private data, and provide application-level protection. Each enclave acts as an independent secure environment. Unlike TrustZone, SGX needs to protect different enclaves, so SGX adopts different isolation protection methods. SGX offers a set of instructions that applications can use to create a private region of memory that is isolated from all other processes, even those with higher privilege levels. Thus, even if a malware or an insider has access to operating system (OS) root privileges, or if the virtual machine manager (VMM) or BIOS are compromised, the SGX-protected application can still operate with integrity and be able to help protect both its code and data.

7.3. Operating Mechanism. The operating mechanism of ARM TrustZone is shown in Figure 11 [36]. The normal world and secure world are two independent environments. A secure service request in normal world contains two procedures:

(1) World state switch, which includes switching from the normal world to the secure world and from the secure world to the normal world.

(2) Execute secure operations: The Monitor module switches to the secure world when the normal world sends a secure service request. After that, the secure world responds to the normal world request, executes secure operations, and returns the result to the normal world.

The operation mechanism of SGX is shown in Figure 12 [36]. Users create enclaves and deploy secure codes and private data in the enclave. SGX protects them from being accessed by external software. Enclave can prove its identity to remote authenticators and provide the necessary functional structure for securely providing keys. Users can also request a unique key, which is unique by combining the enclave's identity with the platform's identity, and can be used to protect keys or data stored outside the enclave. The application requests the secure enclave operation when executing, which needs to set the processor to the enclave model. The processor executes the secure operations and returns the processing results.

Compared to TrustZone, SGX executes enclave switching with fewer costs. Although frequent encryption and decryption steps are involved in SGX, the complete hardware design reduces the encryption and decryption time. However, besides normal, secure operations, TrustZone also executes world status switching and save a large amount of context information, which results in a long time for a single secure request. In [5], the switching cost of TrustZone is tested, which shows that switching from the REE user state to the TEE user state takes 50,000 to 90,000 CPU cycles per operation. While switching from the TEE user state to the REE user state takes 16,000 to 40,000 CPU cycles per operation. The experimental evaluation has been conducted using Hikey 960 development board, 4-core Cortex A73 plus 4-core CortexA53 processor, 4 GB RAM, Linux kernel version is 4.13.0, OP-TEE version is 3.3.0, configuration REE and TEE shared memory size is 2 MB. In [37], the single switching cost of the SGX enclave is tested, which shows that calling an invalid function from an untrusted part to a trusted part takes 7000 CPU cycles per operation, which is about 35 times of the normal system call. The deployed experimental environment is 4-core Intel Core i5-6200U SGX CPU running at 2.4 GHz with 8 GB od 1600 MHz DDR3 RAM. The system type is 64 bit Windows Server 2016 Technical Preview 5. Overall, the single switching cost of the TrustZone world state is about seven times of the single switching cost of the SGX enclave.

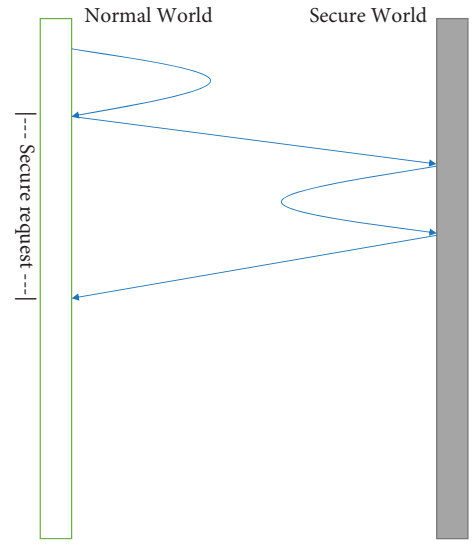


Figure 11: ARM TrustZone secure service request invocation diagram [36].

8. Future Directions

Although TrustZone technology is widely recognized by mobile terminal security applications, but it still faces several security challenges, mainly include the following:

(1) Trusted computing base is too large.

Current TrustZone solutions all contain a powerful security core. Overly complex security cores may contain security vulnerabilities. An adversary can obtain the highest privileges of the system through these vulnerabilities and implement high-intensity

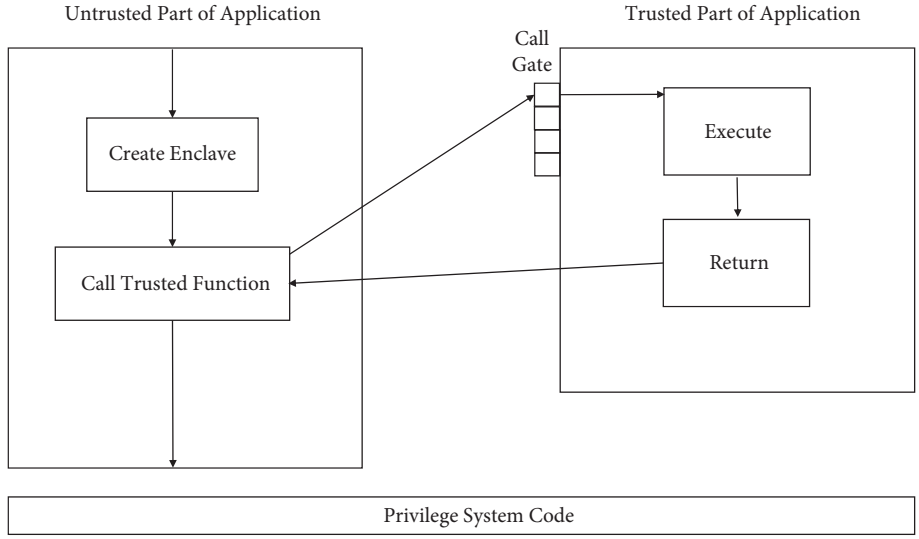


Figure 12: SGX requests the enclave service invocation diagram [36].

attacks. Several Qualcomm QSEE security vulnerabilities have already been exposed, threatening several Android commercial systems. Therefore, how to narrow the system trusted computing base is an important research direction to follow.

(2) Trusted user interaction expenditure is insufficient.

The current TrustZone mainly completes sensitive computing operations such as password calculations, and does not support enough trusted user interactions. This contradicts with the highly personalized mobile applications. Also, subsequent research needs to consider the constraints of low user computing power.

(3) Interaction with common operating systems has security risks.

Currently, TrustZone solutions provide access to security functions for common applications by adding client APIs and TEE drivers to common operating systems. However, the APIs and TEE drivers in ordinary operating systems may be subject to kernel-level malicious code attacks from ordinary systems. In addition, TrustZone cannot do effective identification of applications. This is also the direction that can be further explored.

9. Conclusion

This paper is focused on the system-level security solutions using TrustZone technology proposed by ARM. This article systematically documented the progress and open issues in the field. Specifically, this article discusses four aspects of the technology, i.e., TrustZone technical framework, related research works, advantages and weaknesses with optimization scheme, and comparison with mainstream commercial SGX technique. We have first introduced this technology's hardware and software architecture. Also, its security extension is analyzed in detail. Then, the security mechanism is investigated together with how to realize system-wide security based on hardware and software architecture. The significant achievements of various academic research in the field are summarized. Moreover, an analysis of the advantages and weaknesses of this technology, and the optimization schemes are detailed. Finally, we compared this technology with another mainstream commercial SGX technique and presented some viable future directions.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by Xiamen University Malaysia Research Fund (grant no: XMUMRF/2022-C9/IECE/0033).

References

[1] A. Heidari and M. A. Jabraeil Jamali, "Internet of Things intrusion detection systems: a comprehensive review and future directions," *Cluster Computing*, vol. 10, 2022.

[2] P. Prabhakar, S. Arora, A. Khosla et al., "Cyber security of smart metering infrastructure using median absolute deviation methodology," *Security and Communication Networks*, vol. 2022, Article ID 6200121, 9 pages, 2022.

[3] T. Alves and D. Felton, "ARM trust zone: integrated hardware and software security," *Information Quarterly*, vol. 3, no. 4, 2004.

[4] W. Li, Y. Xia, and H. Chen, "Research on ARM TrustZone," *GetMobile: Mobile Computing & Communications*, vol. 22, no. 3, pp. 17–22, 2019.

[5] Y. Bao-xuan, P. Dong, L. Zhang, and Y. Ding, "Performance optimisation of secure application based on TrustZone," *Computer Engineering & Science*, vol. 42, no. 12, pp. 2141–2150, 2020.

[6] X. Zheng, L. Wen, and D. Meng, "Analysis and research on TrustZone technology," *Chinese Journal of Computers*, vol. 39, no. 9, pp. 1912–1928, 2016.

[7] R. B. Lee, P. C. S. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang, "Architecture for protecting critical secrets in microprocessors," in *Proceedings of the 32nd International Symposium on Computer Architecture (ISCA'05)*, pp. 2–13, Madison, WI, USA, June 2005.

[8] P. Wilson, A. Frey, T. Mihm, D. Kershaw, and T. Alves, "Implementing embedded security on dual-virtual-CPU systems," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 582–591, 2007.

[9] Y. An, B. Zhao, and H. Li, "Extension implementation of TCM in the embedded system based on FPGA," in *Proceedings of the 2013 International Conference on Computer Sciences and Applications*, pp. 749–752, Washington, DC, USA, December 2013.

[10] H. Li and Y. Lan, "A design of trusted operating system based on linux," in *Proceedings of the 2010 International Conference on Electrical and Control Engineering*, pp. 4598–4601, Wuhan, China, June 2010.

[11] A. Ukil, J. Sen, and S. Koilakonda, "Embedded security for internet of things," in *Proceedings of the 2011 2nd National Conference on Emerging Trends and Applications in Computer Science*, pp. 1–6, Shillong, India, March 2011.

[12] A. Khurshid, S. D. Yalew, M. Aslam, S. Raza, and R. Shahid, "TEE-watchdog: mitigating unauthorized activities within trusted execution environments in ARM-based low-power IoT devices," *Security and Communication Networks*, vol. 2022, Article ID 8033799, 21 pages, 2022.

[13] Global Platform Inc, *Global Platform Device Technology TEE System Architecture Version 1.0*, White Paper, New York, NY, USA, 2012.

[14] X. Wang, "The research and application of arm trustzone security isolation technology," M.S. dissertation, University of Electronic Science and Technology of China, Chengdu, China, 2010.

[15] Apple Inc, *White Paper IOS Security*, White Paper, New York, NY, USA, 2014.

[16] SamSung inc, *An Overvire of SamSung KNOX*, White Paper, New York, NY, USA, 2014.

[17] Amd Inc, *AMD Safe Technology White Paper*, White Paper, New York, NY, USA, 2010.

[18] A. M. Azab, "Hypervision across worlds: real-time kernel protection from the ARM TrustZone secure world," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, November 2014.

[19] O. Qingyu, L. Fang, and H. Kai, "High-security system primitive for embedded systems," in *Proceedings of the 2009 International Conference on Multimedia Information Networking and Security*, pp. 319–321, Washington, DC, USA, November 2009.

[20] X. Yan-ling, P. Wei, and Z. Xin-guo, "Design and implementation of secure embedded systems based on trustzone," in *Proceedings of the 2008 International Conference on Embedded Software and Systems*, pp. 136–141, Washington, DC, USA, July 2008.

[21] J. Winter, "Trusted computing building blocks for embedded linux-based ARM trustzone platforms," *STC*, vol. 8, 2008.

[22] K. Dietrich and J. Winter, *Implementation Aspects of Mobile and Embedded Trusted Computing*, Trust Computing, Berlin, Germany, 2009.

[23] W. H. W. Hussin, P. Coulton, and R. Edwards, "Mobile ticketing system employing TrustZone technology," in *Proceedings of the International Conference on Mobile Business (ICMB'05)*, pp. 651–654, Sydney, Australia, July 2005.

[24] M. Pirker and D. Slamanig, "A framework for privacy-preserving mobile payment on security enhanced ARM TrustZone platforms," in *Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 1155–1160, Liverpool, UK, June 2012.

[25] W. Feng, D. Feng, G. Wei, Y. Qin, Q. Zhang, and D. Chang, "TEEM: a user-oriented trusted mobile device for multi-platform security applications," *Computer Science*, vol. 7904, 2013.

[26] X. Hei, W. Gao, Y. Wang, L. Zhu, and W. Ji, "From hardware to operating system: a static measurement method of android system based on TrustZone," *Wireless Communications and Mobile Computing*, vol. 2020, Article ID 8816023, 13 pages, 2020.

[27] Z. Wang, Y. Zhuang, and Z. Yan, "TZ-MRAS: a remote attestation scheme for the mobile terminal based on ARM TrustZone," *Security and Communication Networks*, vol. 2020, Article ID 1756130, 16 pages, 2020.

[28] S. Zhao, "Providing root of trust for ARM TrustZone using on-chip SRAM," *TrustED*, vol. 14, 2014.

[29] L. H. Adnan, Y. M. Yussoff, and H. Hashim, "Secure boot process for wireless sensor node," in *Proceedings of the 2010 International Conference on Computer Applications and Industrial Electronics*, pp. 646–649, Kuala Lumpur, Malaysia, December 2010.

[30] J. González, "A practical hardware-assisted approach to customize trusted boot for mobile devices," *Computer Science*, vol. 8783, 2014.

[31] S.-C. Oh, K. W. Koh, C.-Y. Kim, K. H. Kim, and S. W. Kim, "Acceleration of dual OS virtualization in embedded systems," in *Proceedings of the 2012 7th International Conference on Computing and Convergence Technology (ICCCT)*, pp. 1098–1101, Seoul, Republic of Korea, December 2012.

[32] M. Cereia and I. C. Bertolotti, "Asymmetric virtualization for real-time systems," in *Proceedings of the 2008 IEEE International Symposium on Industrial Electronics*, pp. 1680–1685, Cambridge, UK, July 2008.

[33] D. Sangorrin, S. Honda, and etal, "Dual operating system architecture for real-time embedded systems," in *Proceedings of the 6th International Workshop on Operating System Platforms for Embedded Real-time Applications*, pp. 6–15, Brussels, Belgium, June 2010.

[34] M. Li, Y. Huang, and Y. Liu, "Design of DMA transmission with PCIe bus interface based on FPGA," *Computer Measurement & Control*, vol. 21, no. 1, pp. 233–249, 2013.

[35] V. Anand, J. Saniie, and E. Oruklu, "Threat-adaptive architectures for trusted platform modules in secure computing systems," in *Proceedings of the 2010 IEEE International Conference on Electro/Information Technology*, pp. 1–6, Normal, IL, USA, May 2010.

[36] J. Xia, P. Dong, L. Zhang, Y. Zeng, and Y. Pan, "A comparative study of hardware tee security extension mechanisms in commercial mainstream cpu," *Computer Knowledge and Technology: Academic Edition*, vol. 17, no. 24, p. 6, 2021.

[37] C. C. Zhao, D. Saifuding, and H. L. Tian, "On the performance of intel SGX," in *Proceedings of the 2016 13th Web Information Systems and Applications Conference (WISA)*, pp. 184–187, Wuhan, China, June 2016.