




## Research Article

# TZEAMM: An Efficient and Secure Active Measurement Method Based on TrustZone

Xiaoqing Liu,<sup>1</sup> Yingxu Lai ,<sup>1,2</sup> Jing Liu ,<sup>1,2</sup> and Shiyao Luo <sup>1</sup>

<sup>1</sup>Faculty of Information Technology, Beijing University of Technology, Beijing 100124, China

<sup>2</sup>Engineering Research Center of Intelligent Perception and Autonomous Control, Ministry of Education, Beijing 100124, China

Correspondence should be addressed to Yingxu Lai; [laiyingxu@bjut.edu.cn](mailto:laiyingxu@bjut.edu.cn)

Received 21 July 2022; Revised 25 November 2022; Accepted 4 January 2023; Published 31 January 2023

Academic Editor: Biao Han

Copyright © 2023 Xiaoqing Liu et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

With the rapid development of computer and communication technology, embedded systems are widely used in smart devices. The increasing connectivity of these systems and the difficulties in providing comprehensive security have made such devices vulnerable to malicious attacks. Passive defense technologies and traffic-based intrusion detection technologies are not fully effective against such attacks. Trusted execution environment (TEE) technology can ensure system security against unknown attacks to some extent. Most researchers use TrustZone to implement TEE. However, the problem is that the API interface of the TEE module which provides the service is not secure. Therefore, to actively defend against attacks, we developed a trusted computing active measurement architecture based on TrustZone. To overcome the serious problem that modules in the trusted execution environment need to be passively invoked to provide services, we have proposed an active measurement closed-loop immune mechanism. To reduce the trusted computing base and reduce the performance overhead, we removed certain functional modules from the trusted execution environment. In addition, based on this architecture, we developed a trust chain and dynamic measurement method to ensure the security of the target applications. We changed the traditional attack response method, which requires the entire system to be restarted after an attack, by developing a fallback mechanism that is more suitable for the system. Finally, we verified the effectiveness of the architecture by developing an attack model. Performance testing and analysis showed that the architecture reduced the impact of the security mechanisms on the system. In the future, we will extend our research to more fine-grained measurements.

## 1. Introduction

The rapid development of computing and communications technology has led to the widespread use of embedded systems in smart devices. Given the increasing connectivity of systems and the difficulty of providing comprehensive security, embedded systems are often attacked by malicious parties [1–5]. For example, application-level rootkits typically use code with Trojan characteristics to replace the binary code of a normal application, or they can use other means to modify the behavior of an existing application [6]. Memory can be modified in a number of ways, such as by using a self-modifying code and the *mprotect* call. The self-modifying code requires modification of its own instructions during program runtime [7–9]. It requires the modification

of memory permissions in order to dynamically update the code and data. This method is often used by hackers. In addition, when there is no memory modification permission, an attacker can also gain write access to memory through the *mprotect* call [10], resulting in a loss of integrity. Once memory permissions are modified and the target application's code and data are tampered with or corrupted, the impact is immediate and large. For example, a data leak affects the confidentiality of information, the abnormal operation of devices affects the integrity of information, and system failure affects the availability of information. Therefore, attacks on embedded systems pose a major threat. Passive firewall technologies and traffic-based intrusion detection technologies are not sufficient to defend against the previously mentioned attacks.

Most researchers use the trusted execution environment (TEE) to solve the previous problems. The TEE is designed to provide a secure zone for devices and applications to protect assets and execute a trusted code [11–14]. The TEE is designed to coexist with the rich execution environment (REE). The REE provides a scalable and powerful environment for applications and devices with greater openness and flexibility than the TEE [14] for general purpose computational tasks [15]. The REE can only access the resources of REE and cannot access the resources of the TEE [16]. Therefore, researchers [15, 17–21] often take advantage of the security of the TEE by setting up measurement modules in it. In this way, the resources in the REE are measured regularly to ensure their security.

Many researchers have implemented the TEE using TrustZone, which is the predominant TEE implementation technology on the ARM platform. In this study, we conduct our research based on TrustZone. Dorjmyagmar et al. [18] proposed the integrity measurement architecture TIMA based on TrustZone. TIMA designs periodic kernel measurements and real-time kernel protection components in the TEE to periodically validate the kernel code and data to ensure kernel integrity. Dong et al. [19] proposed the KIMS architecture based on TrustZone. KIMS builds a secure and isolated environment to protect monitoring and measurement modules. Measuring the integrity of the kernel in the REE is achieved by using the privileges of the TEE to determine whether it is destroyed. Dong et al. [20] proposed an active measurement architecture based on the TrustZone design with split-core asynchrony. A trusted encryption module was developed in the TEE to achieve trusted resource monitoring in the REE. Through CPU asynchronous cross-core calls, the problem of excessive CPU time allocation in the TEE is solved, while a storage security method and communication security method are proposed for data protection. The previously mentioned studies propose different resource protection schemes in the REE based on the TEE technique but still have the following limitations:

- (1) The vulnerability of the API interface of the measurement module in the TEE leads to the security monitoring being bypassed. The TEE follows a request-response execution model. The measurement module in the TEE must be invoked by the application in the REE to function properly. First, the client application (CA) in the REE, which is responsible for calling the measurement modules, initiates the security request. Then, the processor is switched to the TEE by calling the secure monitor call (SMC) command, and the trusted application (TA) responsible for the measurement work is executed [22, 23]. If the API interface in the REE to call TA is broken, the measurement work in the TEE will be interrupted and will not provide security for systems and applications such as the Trojan Dvmap virus [24].
- (2) Performance degradation due to active measurement modules working in the TEE. Since active measurements require initiative and regularity, this is usually achieved by improving the priority of the active measurement module and configuring the measurement cycle with high frequency [21]. The active measurement module is deployed in the TEE. When the REE computational task is interrupted by a high-priority measurement, the environment switch is triggered. The resources of the measured target (code segment and data segment) also need to switch environments to obtain. Each environment switch causes a significant performance overhead [25]. Therefore, in the case of a high-frequency measurement cycle, the computational task in REE is severely affected, which also leads to performance degradation of the system.
- (3) The conventional attack response method has the problem of long-time delay and low efficiency. When the active measurement module detects the occurrence of attack behavior, the protection system usually does not design an attack response strategy [26, 27], or it applies the method of restarting the whole system [17] for attack response. When a trusted node in the trust chain is corrupted, the entire system must be rebooted to respond to the attack. The trust chain must also be rebuilt, and the measurement between layers needs to be reformed. This method takes too long and has low recovery efficiency, which affects the work efficiency of the data processing tasks at the REE.

To solve the previously discussed problems simultaneously, this study proposes a TrustZone-based architecture named TZEAMM for trusted computing active measurement, which provides efficient security protection for target applications. TZEAMM includes two execution environments: the rich execution environment for active measurement (REE\_A) and the trusted execution environment for active measurement (TEE\_A). The target applications are deployed in the REE\_A. The functional modules are deployed in the REE\_A and in the kernel, which is a more secure place than the REE\_A due to kernel permission restrictions. The functional modules monitor the target applications through dynamic measurement. The TEE\_A stores trusted reference values and deployed a monitoring module to monitor the functional modules. Within this architecture, we design an active measurement closed-loop based on the trusted chain to protect the functional modules. This solves the problem of the insecure API interface between the CA and TA and ensures that the measurement of the target applications cannot be bypassed. Since the functional modules that perform the measurement operate in REE\_A and the kernel, the target application monitoring runs outside of TEE\_A, mitigating the performance degradation problem and reducing the number of switches between environments by  $2 * n$ . (The  $n$  represents the number of data transmission times of the measured target). In addition, with the previously discussed architecture, the target applications are deployed in REE\_A, the code size of the modules in TEE\_A is smaller, and the trusted computing base (TCB) is smaller, reducing the attack surface [28–31].

We also developed a fallback mechanism that changes the original attack response method of restarting the whole system after an attack.

The main contributions of our study are as follows:

- (i) We develop an active measurement architecture, a trusted chain, and an active measurement mechanism to proactively monitor attacks and protect target applications. We mitigated attacks on the API between CA and TA by ensuring that measurement for target applications cannot be bypassed in REE, thereby mitigating serious attack threats.
- (ii) Removing target applications and functional modules reduces the size of the TCB while ensuring that the target applications are protected in real time. Because the measurements are taken at the REE, the number of environment switches and performance overhead are also reduced.
- (iii) Instead of the conventional method of restarting the entire system when an attack is detected, a fallback mechanism was developed. Using this mechanism decreases the additional performance overhead. It shortens the attack response time and is a more efficient method of attack response.

The remainder of this paper is organized as follows: Section 2 explains the related work, and Section 3 describes the attack model. A detailed introduction to the architecture is presented in Section 4, and Section 5 presents the technology employed in the proposed system. The design idea is described in Section 6, and the implementation method is in Section 7. In Section 8, the experimental analysis is described. In Section 9, we draw conclusions.

## 2. Related Work

In this section, we describe the work related to the technology in this article. In Section 2.1, we introduce SGX and the TrustZone technology. The TrustZone-based defense method is presented in Section 2.2. In Section 2.3, we analyze the performance overhead of the TrustZone-based active measurement method.

*2.1. SGX and TrustZone.* SGX is an Intel-based technology developed to meet the requirements of the trusted computing industry [18, 32–34]. SGX distrusts the OS and implements isolation by building private areas of memory called enclaves. The data in the enclave cannot be accessed from outside and are not visible to outsiders. The application is often divided into two parts: a trusted and an untrusted part [35]. The trusted part of the application is implemented by enclave definition language (EDL) and executed in the enclave to protect the key data and other resources of the application. However, in SGX technology, the resource protection of the enclave is implemented by EDL to define the code. Therefore, it is easy to reverse engineer to spy on the resource data in the enclave.

TrustZone is a hardware architecture [30] designed for embedded devices using ARM to provide a security

framework against device attacks. TrustZone relies on a secure OS, and it achieves isolation by building a secure execution environment. The execution environment is divided into two parts: REE and TEE. Resources that need to be protected can be placed in the TEE and are inaccessible to the REE. If the REE needs data or services from the TEE, it must invoke the application in the TEE. TEE follows the request-response execution model. It executes when it receives a request from REE to a TA. The CA initiates a security request. The processor is then switched to TEE, and the associated TA is executed by invoking the SMC instruction. After executing the TA, the processor switches back to the REE.

### *2.2. TrustZone-Based Protection and Defense Methods.*

According to the TrustZone execution environment classification, there are several main ways to deploy defense technology. One way is to deploy data and applications on the TEE site to achieve isolation protection [36–39]. Guan et al. [36] used TrustZone to provide a lightweight trusted execution environment for security-critical applications, which provides a secure and isolated environment for applications and is more suitable for application execution. It also protects against Iago attacks by developing authentication mechanisms. Sun et al. [37] proposed TrustOTP, which aims to protect one-time passwords (OTP) through the TrustZone security domain and provide reliable OTP generation and trusted OTP display. Salman and Du [38] use TrustZone's TEE to ensure the integrity of data collected from peripherals and the security of data processing, guaranteeing QR code payments and location verification services in mobile phones. However, this method cannot protect the system from DoS attacks. Li et al. [39] implement a verifiable mobile advertising framework AdAttester based on TrustZone. AdAttester uses TrustZone's TEE to collect and authenticate the two primitives of unforgeable clicks and verifiable displays to accurately distinguish AD fraud from legitimate AD operations. However, as the number of applications on the TEE increases, the previous method [36–39] will increase the TCB [28, 29], and a larger TCB makes the system more vulnerable [30]. As shown in Table 1, the previously discussed methods have increased the scale of TCB to some extent. Moreover, some research needs to complete the system call in REE, which is a security risk. Our method not only reduces the TCB but also guarantees the security of the application.

Another method is to run a measurement module in TEE to protect applications and data deployed in REE [15, 17, 26, 27]. By calculating the hash value of the measured target and comparing it with the reference value, this method can determine whether the resource is under attack. Jia et al. [15] proposed an active trusted computing model to realize active measurement, design security computing units in the TEE, and realize trusted boot and dynamic monitoring of systems and applications in the REE. However, this approach does not use semantic constraints to describe the

TABLE 1: Comparison of different defense methods in classification 1.

Methods	Character
TrustShadow [36]	TCB: general and security: low
TrustOTP [37]	TCB: larger and security: high
Reference [38]	TCB: general and security: low
AdAttester [39]	TCB: larger and security: high
TZEAMM	TCB: smaller and security: high

semantic integrity of the kernel’s dynamic data. Tian et al. [17] constructed a security-first model, designed a trusted monitoring and measurement module in the TEE, and performed fine-grained dynamic monitoring of the kernel module using semantic reconstruction to protect the integrity of the kernel. However, in this method, the measurement is performed on the TEE, and the acquisition of measurement data is subject to multiple environment switches, which affects the performance of the system. Ling et al. [26] proposed a paging-based process integrity measurement and proof method for IoT devices based on TrustZone. TEE periodically measures the code segments of the process in REE to ensure the runtime integrity of the REE. However, this method cannot protect against attacks on the API interfaces between CA and TA. Yalaw et al. [27] designed TruAPP, which designed verification components in the TEE, and verified the integrity of other security components and applications such as trackers in the REE while providing authenticity verification features such as static watermarking and hash encryption. However, each time the method provides the authenticity verification function, it needs to authenticate component integrity and system integrity including the entire kernel, which affects the timeliness of the application function. As shown in Table 2, the previously discussed methods all have the problem of unsafe API interfaces. On the basis of protecting the application security, our scheme protects the security of the measurement service API interface and has a small TCB.

In addition, there are several research methods that take advantage of the TrustZone architecture. Sun et al. [29] designed an isolated computing environment in REE to reduce the size of TCB while protecting the secure code and used the controller in TEE to realize inter-domain switching and isolation. Ge et al. and Wang et al. [40, 41] realized the introspection mechanism through the method of probes. When there was an operation to update the page table, an exception was thrown and the processor switched to the TEE for security verification. Jiang et al. [22] implemented CA identity authentication in the TEE to prevent sensitive data leakage and DoS attacks given the security threats between CA and TA sessions. As shown in Figure 1, among the research methods with different characteristics, our method has the advantages of small TCB, high security, low-performance overhead, and deployment of efficient attack response methods.

*2.3. Performance Overhead under the TrustZone-Based Active Measurement Method.* When the TA in the TEE measures the CA in the REE, it introduces some performance

TABLE 2: Comparison of different defense methods in classification 2.

Method	Character
Reference [15]	TCB:general/larger
Reference [17]	Application: safe
Reference [26]	API interfaces for measuring services: unsafe
TruAPP [27]	TCB:smaller
TZEAMM	Application: safe
	API interfaces for measuring services: safe

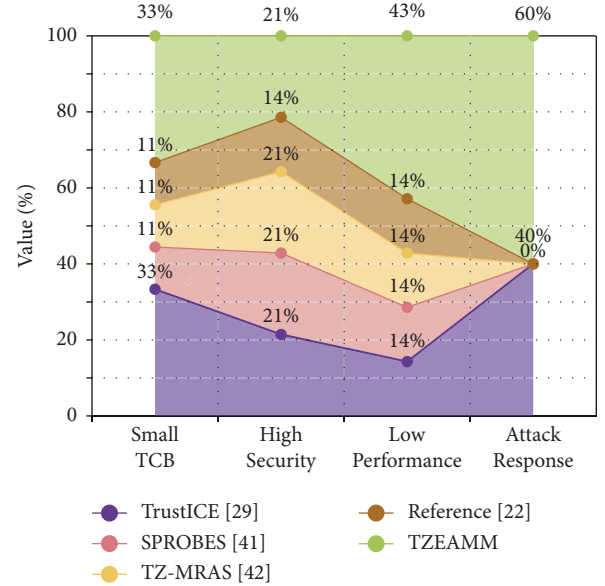


FIGURE 1: Comparison of different defense methods in classification 3.

overhead that is reflected in the cost of environment switching, shared memory allocation, shared memory deallocation, and so on. First, before executing the TA, part of the memory area of REE is reserved as shared memory. The system copies the data to be passed to the TA and stores it in shared memory. Then, when the CA calls the TA, the environment is changed. However, there are times when the TA responds to an interrupt from the REE and returns to the TA after the interrupt is completed. The number of environment switches and the performance overhead then increase. After the TA executes, the data returned to the CA are copied to shared memory. When the environment switches back to the REE, the CA acquires the data from shared memory. Finally, the REE releases the shared memory. Therefore, running applications based on TrustZone incurs significant additional performance overhead. According to one analysis [25], loading and executing TA, environment switching, and allocating and releasing shared memory all incur some performance costs. Therefore, to improve performance, we should reduce the frequency of these factors as much as possible. Our work reduced the performance overhead of factors such as the number of world switches by deploying most of the functional modules outside of TEE.

### 3. Attack Model

Some researchers deploy functional modules on the TEE to dynamically measure and assess the health of target applications in real time and assess their state. Functional modules assess the security state of the application code and data segments to protect them in real time. However, when using the TrustZone mechanism, the modules in the TEE, which provide security services, are passively invoked by applications in the REE. The security services will be deactivated whenever the API interface for these passive calls is deleted, such as the Trojan Dvmap virus [24].

As shown in Figure 2, Target Apps represents the protected applications. The collection of components in the TEE that provide measurement services to the Target Apps is called functional modules. The functional modules' invoker, which calls functional modules, and the functional modules periodically perform their measurements when invoked by this module. The specific attack process is as follows.

By modifying the entire page table, the attacker alters the properties of the memory page so that it is writable. The contents of the memory page corresponding to code and data segments can be modified. First, the attacker manipulates the functional modules' invoker to break the API call interface and bypass the measurement service of the functional modules. When the invocation interface of the measurement service is disrupted, the attacker destroys the Target Apps and accesses the apps' privacy data.

In summary, we assume that TEE is secure, and complex hardware attacks such as side-channel attacks are beyond the scope of this article. We only consider the following two points of attack: one is the modules and applications running on REE. The other is the API interface in REE, which is used to invoke the TEE security service. We only consider the security of the process code and data segments.

### 4. System Overview

In this study, a trusted computing active measurement architecture based on the TrustZone architecture named TZEAMM is proposed and simulated with an open portable trusted execution environment (OP-TEE). TZEAMM is based on two execution environments, REE\_A and TEE\_A, as shown in Figure 3. REE\_A performs normal computational tasks and provides measurement protection for the target applications. TEE\_A stores trusted reference values and checks the security of functional modules in REE\_A.

TZEAMM consists of four main modules and seven submodules. As shown in Figure 3, the main modules are the CA, TAOC (trusted application on the client), TA, and FM-K (functional module-kernel) modules. The target applications running in the CA module perform the computational tasks. The TAOC module is a functional module running in REE\_A. It contains the submodules removed from TEE\_A, the control module, and the assessment module. The control module schedules the entire measurement process in an active mode. The assessment module determines whether the measurement result is as expected. The TA is a functional module deployed in TEE\_A and

includes the TBDB (trusted base database) submodules and the monitoring module. The TBDB stores trusted reference values of modules and applications. The monitoring module ensures the security of the functional modules moved into the kernel. The FM-K module also consists of submodules: the measurement module and the communication module. The measurement module is the core module that implements active measurement mechanisms and measurement strategies. The communication module transmits information between the kernel and the user area. In addition, the OP-TEE message transmission submodule in the kernel is used to transmit information between REE\_A and TEE\_A.

In this architecture, the functional modules are deployed in REE\_A, and the measurement data acquisition mainly depends on the inter-module communication and the kernel-user communication. CPU mainly operates in REE\_A, which has little impact on the target application. The performance cost caused by environment switching times is also greatly reduced.

### 5. Innovative Technology Employed in the Proposed System

In this section, we present two innovative technologies based on the proposed system: an active measurement mechanism for active defense and a fallback mechanism for attack response.

*5.1. Active Measurement Mechanism.* As shown in Figure 4, the active measurement mechanism includes two parts: the active measurement of the target application and the internal immunity of the active measurement modules realized by the active measurement.

One is the active measurement of the target application by active measurement modules. The active measurement modules consist of TAOC, FM-K, and monitoring modules. Active measurement does not require the invocation of a target application. During the operation of the target application, the active measurement modules measure the target application according to the security policy to ensure that the behavior of the target application is consistent with expectations.

The other is the active measurement closed-loop immunity of the active measurement modules. To improve the operating efficiency of the security mechanism, the TAOC and FM-K modules are executed in REE\_A. They may be threatened by the attack described in Section 3. The attack on the TAOC and FM-K modules will affect the normal operation of the active measurement mechanism. Therefore, we have developed an active measurement closed-loop immune mechanism to ensure the security of the active measurement modules. Based on the trust transfer of three functional modules, the active measurement closed-loop immune mechanism is implemented, as shown in Figure 4. The monitoring module runs in TEE\_A and is protected by TEE\_A. The monitoring module, as the starting point for trust transfer, actively measures the FM-K module. The trusted FM-K module actively measures the TAOC module,

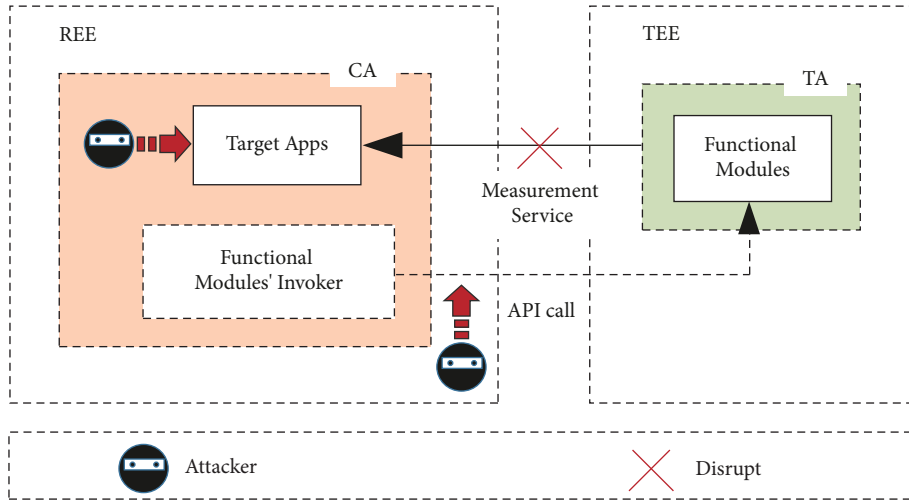


FIGURE 2: Schematic of the attack model.

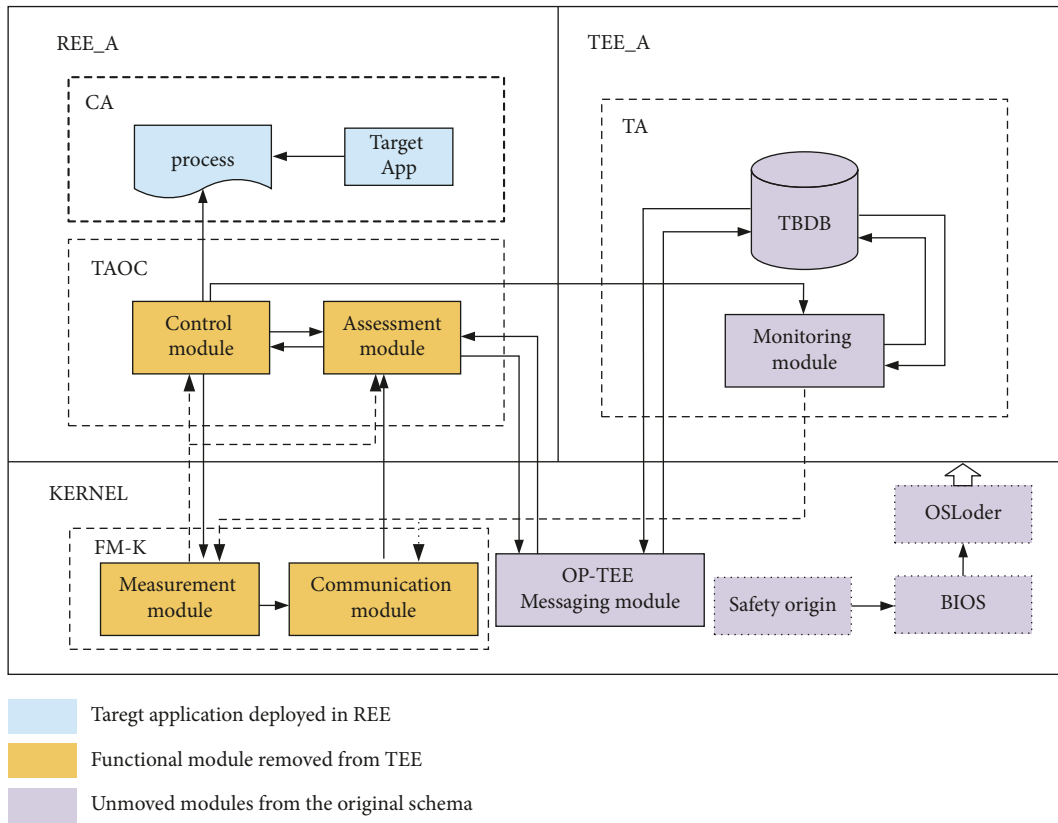


FIGURE 3: System architecture diagram.

which is at the end of the trust transfer. Based on the abovementioned trust transfer, the TAOC module is provided with a function to control and call the monitoring module. In this way, an active measurement closed-loop is established. Under the supervision of the FM-K module, the caller of the monitoring module, namely the TAOC module, controls the monitoring module's measurement task to solve the problem of the monitoring module's unsafe passive call interface. Each functional module is controlled by the superior module in the closed-loop. Even if it is attacked by the

passive call cancellation attack described in the attack model, it can be detected by the superior module and return to the normal state through the fallback mechanism (described in Section 5.2). By developing an active measurement closed-loop immunity mechanism, we solve the problem of an insecure API interface between TA and CA, protect the monitoring module in TEE\_A, and then ensure the security of the external functional module of TEE\_A.

Consequently, the active measurement mechanism ensures the security of the target applications with the security

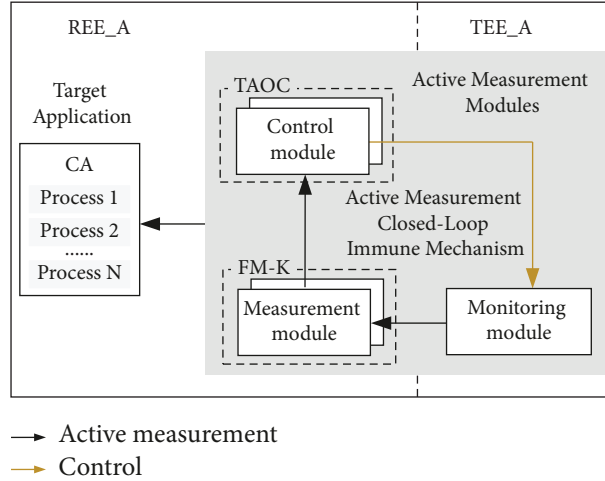


FIGURE 4: The active measurement mechanism model in TZEAMM.

functional module, which can effectively protect all code and data segments from tampering.

**5.2. Fallback Mechanism.** In the face of attacks on the active measurement modules and target applications, the fallback mechanism is employed to fall back to the nearest trusted module. Trust is restored from this module. The module is then given permission to restart the attacked module and transfers the trust down until the security of the entire system is restored. As follows, attacks may occur in two places.

- (1) FM-K is the measurement and communication modules. When an attack occurs in the FM-K module, the right of control falls back from the TAOC module to the monitoring module in TEE\_A, as shown in Figure 5. The monitoring module automatically restores and measures the attacked module through a backup. The chain of trust is then retransmitted down, starting with the FM-K module. Finally, the TAOC module, validated as a trusted component, is used to regain control of the application process.
- (2) TAOC, i.e., the control and assessment modules. When an attack is detected in the TAOC module, it relinquishes control to the higher-level FM-K module and stops measuring the application process, as shown in Figure 6. The untrusted TAOC module is destroyed and restarted by the FM-K module in the user mode. It then resumes measurement and control of the application process after being found to be trusted.

This method replaces the previous way of restarting the entire system as soon as an attack is detected, and it eliminates to some extent the additional performance burden.

## 6. Design

In this section, we describe the processes of static measurement and dynamic measurement, as well as the data interaction description method used in the dynamic measurement.

### 6.1. Data Interaction Description Method

*Definition 1.* The measured entity can be expressed as follows:

$$EM = \langle \text{target, identifier, state, data} \rangle. \quad (1)$$

Here, the target is the measured target, which may be the user mode application process (ClientAP) or a functional module that has been removed from the TEE (FM). The identifier is a unique identifier of the measured target, referred to as the process number (*PID*) or the FM-K module name (*kernel\_symbol*). The state represents the status of the target, i.e., whether it is trusted, untrusted, or null. Null here denotes that the status of the target has not been measured yet. Data are the transmission data and can be the measurement result of the measurement module, the trusted reference value from the TBDB, or null.

*Definition 2.* The transmission of the measured target information can be expressed as follows:

$$S \triangleright_1 \{(\text{control, measurement, } EM) \cap EM = \langle \text{target, identifier, null, } \emptyset \rangle\} @t. \quad (2)$$

This means that the control module sends information about the measurement target to the measurement module

within  $t$  seconds. Here, the measured entity  $EM$  in Definition 1 gives specific information about the target. When the state

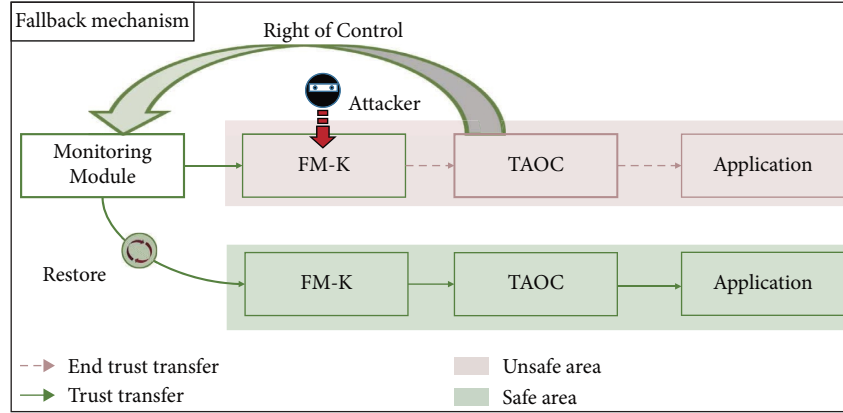


FIGURE 5: Fallback mechanism in case 1.

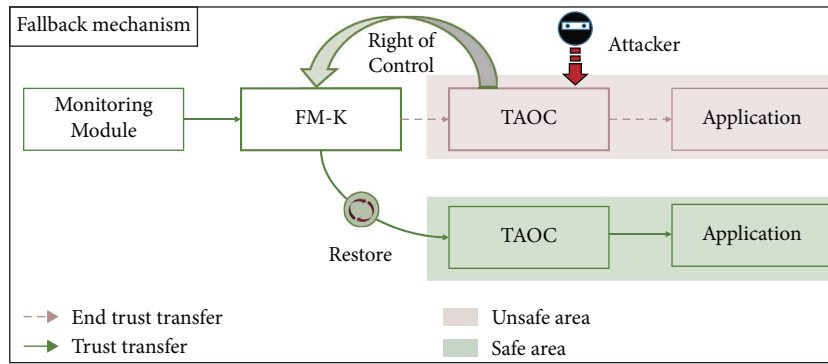


FIGURE 6: Fallback mechanism in case 2.

of the measured target is null, the control module outputs the measurement work for the measurement goal. At this stage, the data are represented by  $\emptyset$ .

*Definition 3.* The transfer of the measurement value can be denoted as follows:

$$S \triangleright_2 \{(\text{measurement, communication, EM}) \cap \text{EM} = \langle \text{target, identifier, null, hash} \rangle\}. \quad (3)$$

This indicates the transmission of the measurement value from the measurement module to the communication module. Again, the state of the measured target is set to null.

The hash value calculated by the measurement module is stored in the data.

In the same way as before,

$$S \triangleright_3 \{(\text{communication, assessment, EM}) \cap \text{EM} = \langle \text{target, identifier, null, hash} \rangle\}, \quad (4)$$

is expressed as the flow of the measurement value from the communication module to the assessment module.

*Definition 4.* The transfer of the trusted reference value can be described as follows:

$$S \triangleright_4 \{(\text{TBDB, assessment, EM}) \cap \text{EM} = \langle \text{target, identifier, null, hash\_base} \rangle\}. \quad (5)$$

The formula represents the transfer of the trusted reference value from the TBDB to the assessment module. The

state of the target is null here as well. The data are set to the trusted reference value in the TBDB.



*Definition 5.* The transmission of the decision result indication can be expressed as follows:

$$S \triangleright_5 \{(\text{assessment, control, EM}) \cap \text{EM} = \langle \text{target, identifier, trusted|untrusted, } \emptyset \rangle\}. \quad (6)$$

This indicates the transfer of the state of the measured target from the assessment module to the control module, whose value is trusted or untrusted.

*6.2. Static Measurement.* This study is grounded in the TrustZone architecture's security mechanism, which can ensure the integrity of the REE\_A application start-up and load times. The trust chain can be divided into two stages. The first stage is the trust verification from the firmware start-up to the loading of TEE\_A. The second stage is loaded by TEE\_A to verify the trust of REE\_A systems and applications. In this study, the TrustZone architecture is simulated using OP-TEE.

The trusted root of the system is code in read-only memory (ROM) in the CPU kernel, which is considered as trusted in this study, and is not tampered with. This code is loaded into the CPU's static random-access memory (SRAM) during start-up. Based on the PUF characteristic of the initial power-up value of the CPU on-chip SRAM [20], the data in the SRAM are overwritten by other data after the system starts. The rest of the system is verified by the trusted chain to ensure the security of the entire system.

The first step of the first stage is to use the on-chip trusted code segment to verify the security of the basic input and output system (BIOS). After passing the verification, the BIOS is loaded and the trust is passed to the segment. The trust is finally transferred to TEE\_A via the BIOS and OSLoader. The TEE\_A boots after verification, and the first stage of authentication ends [42].

The second stage of authentication starts with the trusted TEE\_A. First, TEE\_A validates the operating system of REE\_A through the OP-TEE mechanism. REE\_A is activated when TEE\_A finds it trustworthy. Next, the monitoring module in TEE\_A determines whether the FM-K module is trusted. After the FM-K module is determined to be trusted, it loads and measures the TAOC module. Finally, the active measurement modules measure the application launched in REE\_A. Here, certification is a coarse-grained measurement of the code segment and data segment. The loaded target can be executed if it expects the measurement value. Thus, the trust chain extends from the trusted root to the entire system.

In the following sections, we describe the protection methods for code segments. The protection methods for data segments are the same as those for code segments and therefore do not need to be discussed separately.

*6.3. Dynamic Measurement.* If you rely only on static measurements, the safety of the system is not guaranteed. For example, static measurements are not able to handle TOU-TOC (time of use-time of check) attacks [21, 41, 43], where the process remains unaffected before the

measurement but is destroyed afterwards. Only with the help of dynamic measurement and the definition of an appropriate measurement period can we ensure the security of the system during runtime. The object of the dynamic measurement includes two parts: the active measurement modules in REE\_A and the target application in REE\_A.

The dynamic measurement of the application is performed by the active measurement modules. Therefore, it is important to ensure the security of these modules. Once they are attacked, the measurement results they provide are no longer trustworthy. As shown in Figure 7, the monitoring module in TA periodically measures the FM-K module to ensure its security. The following describes the measurement procedure for the code segment. First, the monitoring module receives the entity information of the FM-K module, expressed as  $\text{EM} = \langle \text{FM, kernel\_symbol, null, null} \rangle$ . Second, the monitoring module accesses the kernel symbol list to determine the address of the code segment. Finally, the code segment is measured and evaluated. The FM-K module, which is trusted, authenticates the TAOC module. Here, the FM-K module collects information about the TAOC module, which is represented as  $\text{EM} = \langle \text{FM, pid, null, null} \rangle$ . Then, the code segment of the module is acquired according to the PID. Finally, the FM-K module measures the code. If the TAOC module and the FM-K module are found to be untrusted, the fallback mechanism is used to recover them.

The active measurement modules that are verified to be trusted measure the application to ensure the security of its runtime. As shown in Figure 8, the control module regularly sends the measurement module an authentication request for the process in the CA module as  $S \triangleright_1 \{(\text{control, measurement, EM}) \cap \text{EM} = \langle \text{clientAP, pid, null, } \emptyset \rangle\} @t$ . After receiving the process unique identifier PID, the measurement module receives the code segment of the corresponding process space and performs the hash operation on it. Then, the measurement module sends the calculated hash value to the communication module. This is expressed as  $S \triangleright_2 \{(\text{measurement, communication, EM}) \cap \text{EM} = \langle \text{clientAP, pid, null, hash} \rangle\}$ . According to the interaction strategy between the kernel and the user state, the communication module transmits the hash value to the assessment module in the user mode, described as  $S \triangleright_3 \{(\text{communication, assessment, EM}) \cap \text{EM} = \langle \text{clientAP, pid, null, hash} \rangle\}$ . When the control strategy is executed, the assessment module retrieves the trusted reference value from the TBDB in TEE\_A. The data are passed between REE\_A and TEE\_A via the OP-TEE message transmission module. After receiving the communication request from the assessment module to the TBDB, the SMC uses the instruction to switch the environment. Then, it sends a read request to the TBDB. The TBDB stores the trusted reference value in the shared memory, and the environment switches back to REE\_A. This process is referred to as  $S \triangleright_4 \{(\text{TBDB, assessment, EM}) \cap \text{EM} = \langle \text{clientAP, pid, null, hash\_base} \rangle\}$ . The assessment module retrieves the trusted reference value from the shared memory, compares it with the hash value provided by the measurement module, and sends the comparison result to the control module. The previously described process can be expressed as  $S \triangleright_5 \{(\text{assessment, control, EM}) \cap \text{EM} = \langle \text{clientAP, pid,$

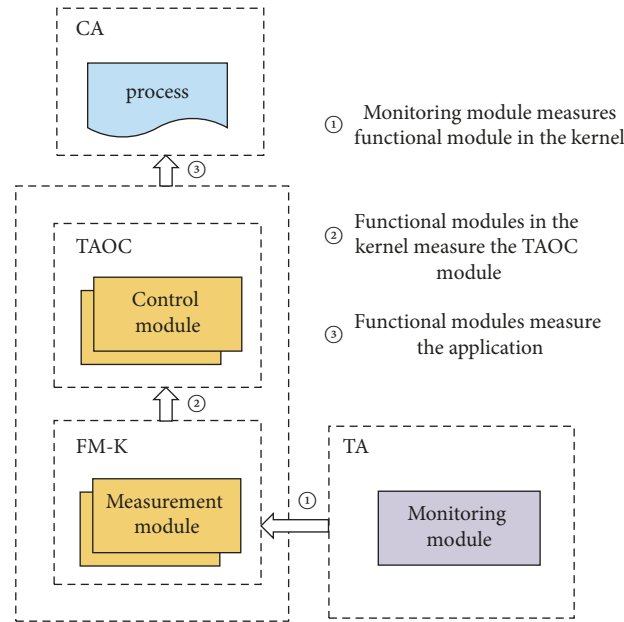


FIGURE 7: The schematic diagram of dynamic measurement.

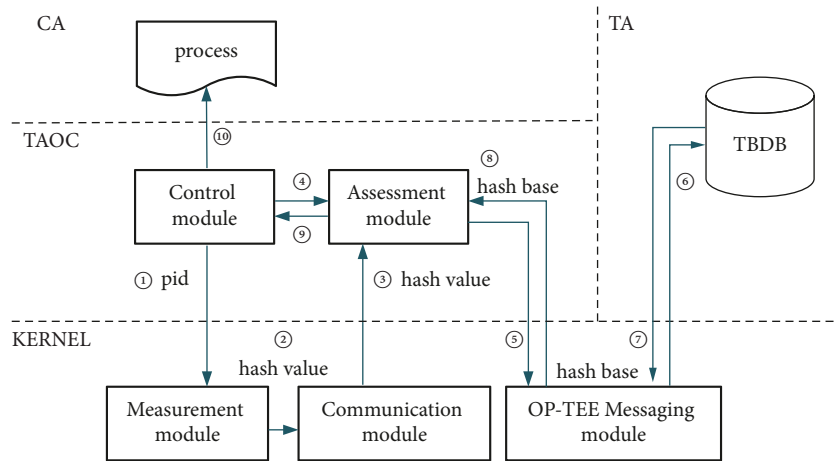


FIGURE 8: Dynamic measurement details.

trusted|untrusted,  $\emptyset$ }. Finally, the control module applies the appropriate strategy to the process, depending on the outcome of the decision. If the result is credible, the module does not interfere with the normal operation of the process; otherwise, it forcibly terminates the process. This completes the dynamic measurement of the process.

## 7. Implementation

In this section, we explain the experimental environment and implementation methods of the key technologies.

**7.1. Execution Environment.** The quick emulator (QEMU) 2.9.50 of Ubuntu 16.04 and OP-TEE 2.6.0 was used in this experiment. The two prototype systems were virtualized by QEMU. TZEAMM was built so that REE\_A and TEE\_A could run simultaneously. The virtual REE\_A ran on a

complete Linux system. It could obtain the security services of TEE\_A by calling the API provided by OP-TEE [44]. We used OP-TEE to simulate the trusted execution environment of TrustZone.

In this study, the Netlink communication mechanism used the socket application programming interface (API) for interprocess communication, which enables full-duplex and asynchronous communication. It also enables data transmission between the user space and the kernel space. It provides not only a set of standard API interfaces for user-space programs but also a set of special APIs for kernel modules. By calling the API, messages can be passed between the two spaces.

The following describes how the CA invokes the services in the TA. First, the CA calls the TEE\_Client\_API to initiate the system call to enter the kernel state in REE\_A and then locates the appropriate driver based on the parameters called. The driver invokes the SMC command, switches the

processor to the secure kernel state, and enters the security environment of TEE\_A. Then, the service request of CA is sent to TEE\_A. TEE\_A calls the corresponding TA via the TEE\_Internal\_API and transmits the data that needs to be transferred to the CA. Finally, it goes back to the TEE\_A kernel state and invokes the SMC command to switch back to REE\_A.

**7.2. Implementation Method.** The API interface is used by the control module to control the operation of the monitoring module and by the assessment module to read the trusted reference value from the TBDB. In the context of the OP-TEE mechanism, the implementation of the API interface of REE\_A that calls the TEE\_A service is as follows: to begin, the *TEEC\_InitializeContext* function needs to initialize and set up a context with TEE\_A. Then, the *TEEC\_C\_OpenSession* function is applied to establish the session window between the CA and the TA. The CA sends an execution request to the TA by the *TEEC\_InvokeCommand* function to perform the specific operation. Finally, the session window is closed using the *TEEC\_CloseSession* function and the established context is cleared by the *TEEC\_FinalizeContext* function. When REE\_A calls the API interface of TEE\_A, the TA, such as the TBDB and the monitoring module, must perform the following steps: to facilitate the API call, an entry point for the CA is set up by executing the *TA\_CreateEntryPoint* function to facilitate the API call. Then, the *TA\_OpenSessionEntryPoint* function is used to establish a communication channel between the CA and the TA. The *TA\_InvokeCommandEntryPoint* function is executed to receive the instructions and parameters transmitted by the CA, and the function is implemented according to the command ID. The communication channel is closed by executing the *TA\_CloseSessionEntryPoint* function. The access point is finally removed by executing the *TA\_DestroyEntryPoint* function.

To send the PID of the measured process to the measurement module, it is necessary to establish a communication connection between the user state and the kernel state. To do this, the user-mode process needs to create a socket and use the *bind* function to bind the address used to the socket. Then, the function *ioctl* is used to transfer the PID to the measurement module's driver file, and the function *recvfrom* is used to receive the returned value from the kernel.

The measurement-specific process for the code segment is described as follows: the task structure of the process is determined by calling the function *find\_vpid* according to the PID. The address of the code segment is obtained, and the code is mapped to the preallocated memory. When visiting the preallocated space, the SHA cryptographic algorithm is used to hash the code segment and obtain the measurement value. The measurement method for TAOC is the same as for the target application. The FM-K module's code segment acquisition method is described as follows: the kernel module can be accessed in the form of a kernel symbol. Therefore, the function *kallsyms\_lookup\_name* is used to read the kernel symbol list and obtain the address of

the code segment. Then, the code segment obtained by the access address is stored in a variable of the *TEEC\_Operation* data type for transmission to the monitoring module in TEE\_A.

## 8. Evaluation

In this section, we analyze and evaluate the proposed system from two aspects: function and performance.

**8.1. Function Evaluation.** In this section, we examine the functions in two ways: in Section 8.1.1, we verify the functions through attack tests. In Section 8.1.2, we theoretically analyze the functions included in this system by comparing and analyzing them against the other three TEE designs and integrity measurement schemes.

**8.1.1. Function Verification.** We implemented the attack model to test whether TZEAMM can detect and respond to the attack. The attack module resides in the kernel, and its purpose is to attack and destroy the process by modifying its code segment. It looks up the process descriptor of the target process based on the PID in the process descriptor list. By modifying the flag bit of the virtual memory region corresponding to the process, the region changes from a read-only property to a read-write property. The code segment in the area was modified to perform the attack on the specified process. After the attack started, the measurement value process changed. The assessment module showed that the comparison of the measurement value did not match, thus detecting an attack on the code segment of the process. Finally, the control module automatically resumed the attack process to prevent the system from being further compromised. Through functional tests, we verify that the defense mechanism can effectively defend against code segment attacks such as CSE509-Rootkit, Reptile, and enyelkm by setting corresponding measurement targets.

**8.1.2. Comparison and Analysis.** The study's scheme is compared with the other three TEE runtime systems. The results are shown in Table 3.

iFlask is a Flask scheme based on TrustZone proposed by Zhang and You [45] which is an access control module integrity protection system. It uses the TEE to protect the access control security server subsystem, and it not only implements memory protection policies but also provides an exception trap mechanism to protect other functional components. The proposed memory protection strategy extends the scope of TrustZone's protection while restricting access to physical pages. The integrity of the process code segment, all functional components, and applications in REE is protected by the method described previously. In addition, the attack response technique is implemented through the exception trap mechanism, which hooks the exception modification and sends it to the TEE for a response. However, the impact of a higher TCB is ignored, and the only defense strategy is a passive access control-based one.

TABLE 3: Function comparison of this system with other runtime systems.

Function	iFlask [45]	TLR [46]	TAAuth [47]	TZEAMM
Small TCB				
Process code segment integrity				
Attack response				
Protection of functional components				
App protection				
Active defense				

: Realization. : Partial-realization. : No-realization.

TLR [46] is a trusted language runtime approach that allows developers to create trusted components using high-level languages. The computation portion of a sensitive application is split into the TEE, while the rest remains in the REE. Isolated runtime environments for distinct applications are created in the TEE, and functional modules are set in the REE to connect the two parts of the application. This reduces the TCB while protecting the critical computational activities of the application. Attackers can still cancel the call to the TEE service by modifying code segments or compromising applications and functional modules in the REE due to the lack of memory monitoring and protection in the REE. As a result, the TEE technique, which only covers the key operation part of the program, is not sufficient to ensure the security of the entire application. The integrity of the process code segments, functional components, and applications of REE is not adequately protected. The performance problems caused by frequent invocation of the sensitive computing component of the TEE when the environment switches are not discussed in this article nor are the response to attacks and active defense.

TAAuth [47] is a TrustZone-based technique for protecting the security of authentication applications. The isolated environment is configured in the REE and protected by a memory isolation mechanism. In this method, the critical components of the application are isolated in the REE, and the TEE is used to switch between the normal and the isolated area in the REE. As a result, this technique effectively minimizes the TCB's size. Using a memory isolation method also ensures the security of functional components. However, similar to the TLR, the isolation area can only ensure the security of the internal parts and not the rest of the application outside the isolation region. Therefore, the integrity of the application component outside the isolation region, as well as the associated code segment, cannot be guaranteed. Attack response and active defense are not mentioned in the article nor are the performance issues caused by frequent environment switching, which are still worthy of attention.

To reduce the TCB, we excluded the majority of functional modules from the TEE in our research. To ensure the security of removed functional modules, we set up the monitoring module in the TEE. To cope with the attack mode specified by the attack model, we developed an active measurement mechanism. To ensure the integrity of the

application and process code segments, the active measurement modules dynamically measured the target applications. The attack response mechanism, based on the fallback mechanism, and the active defense method, based on dynamic measures, work together to protect the system as much as possible.

Next, we performed a comparative analysis with four TrustZone-based integrity measurement scheme methods. The results are shown in Table 4. KIMS [19] is a scheme to provide measurement integrity protection for the kernel in REE. TZTCM [20] is an active measurement scheme for applications. TruAPP [27] is a scheme to provide integrity verification for functional modules and applications such as tracers in REE. Reference [26] is a scheme for measuring application code segments. Our solution provides a dynamic measurement for applications and functional modules. Regarding the size of TCB, KIMS, and TZTCM, the authors in [26] do not consider the reduction of the size of TCB and the attack surface that TCB provides. TruAPP has removed functional modules such as the tracker, and the TCB is reduced to some extent compared to other schemes. In our scheme, most of the functional modules have been removed and the TCB is smaller. Our scheme protected the API interface between CA and TA. However, KIMS, TZTCM, and TruAPP disregarded the security of the API interface while providing services in functional modules. The measurement module in Reference [26] is called by the Attestation CA in REE to run but ignores the situation that the security measurement service is bypassed after the API interface of the Attestation CA call is attacked. For attack response, TZTCM proposed a shutdown restart or alert scheme. We propose a more efficient fallback mechanism; other schemes do not consider the response actions after an attack.

In addition, we also made a quantitative comparison among the abovementioned schemes. Since the system architectures of the abovementioned solutions are different, the details of the interaction between modules are not covered in the articles. Therefore, we compare the parameters that have a great impact on performance, that is, the environment switch. The measurement modules of other schemes such as KIMS are deployed in TEE, so it needs to switch the environment  $2 * n$  times to obtain the measurement data, and 2 times environment switches are needed to call and return the measurement results. However,

TABLE 4: Function comparison of this system with other integrity measurement schemes.

Models	KIMS [19]	TZTCM [20]	TruAPP [27]	Reference [26]	TZEAMM
Integrity protection	Realize	Realize	Realize	Realize	Realize
TCB	Larger	Larger	General	Larger	Smaller
Protection of CA-TA API interfaces	None	None	None	None	Realize
Attack response	Shutdown etc	None	None	None	Fallback
Number of environment switches	$2 * (1 + n^1)$	$2 * (1 + n^1)$	$2 * (1 + n^1)$	$2 * (1 + n^1)$	2

<sup>1</sup> $n$ : if the size of the code segment is  $M$  and the size of the shared memory used for data transfer is  $Mg$ , it must go through  $2 * n = 2 * \lceil M/Mg \rceil$  environment switches.

TZEAMM deploys the measurement module in the kernel of REE and TBDB in TEE; only two environment switches are needed to obtain the trusted reference value. Therefore, TZEAMM optimizes the performance.

**8.2. Performance Evaluation.** In this section, we analyze performance mainly from two perspectives: Section 8.2.1 analyzes the time overhead caused by the communication between REE\_A and TEE\_A, and Section 8.2.2 examines the performance of the system using the performance test tool, Lmbench.

**8.2.1. Performance Optimization Analysis.** In the raw integrity measurement architecture, when TA is invoked by CA, there is a time cost due to environment switching, registration, and shared memory release. As shown in Figure 9, when the CA invokes the service in the TEE, an environment switch occurs. In addition, the shared memory must be registered in the REE. The control module sends the control strategy to the measurement module, which requires communication between the modules. The measurement module must access the REE kernel to obtain the code segment, which requires an environment switch and communication between the user mode and the kernel state. If the size of the code segment is  $M$  and the size of the shared memory used for data transfer is  $Mg$ , it must go through  $2 * n = 2 * \lceil M/Mg \rceil$  environment switches [48]. After the measurement is completed, the following communication between the modules is required: transferring the measured value, retrieving the trusted reference value, and sending the measurement result to the control module. The control module executes the appropriate strategy according to the decision result. Then, it must release the shared memory, switch the environment, and return to the REE. In this process, we have to go through a total of  $2 * (1 + n)$  environment switches, one shared memory registration, one shared memory destruction, five communication processes between modules, and two communication processes between the kernel mode and the user mode. Since most of the measurement work is carried out in the TEE, if the TA requires a large amount of computation and takes up the CPU for a long time, it would shield the interrupt requests from the REE. This causes the process in the REE to starve due to a long wait time (process hunger occurs when the waiting time has a significant impact on process progress and response. Starvation death occurs when the process of starvation, in a sense, waits until it has no practical meaning,

even when it is completed). To reduce the phenomenon of “starve to death,” the priority of the processes in the REE must be increased and some of the REE interrupts must be answered during the TA operation. Each interrupt response introduces an overhead of at least two environment switches, which greatly affects the performance of the system.

In the mechanism proposed in TZEAMM, the following additional costs are incurred when making measurements, as shown in Figure 10: the inter-module communication cost incurred, which is caused when a measurement request is sent to the measured process, the cost incurred when the PID is transmitted to the control module, the cost incurred when the measurement value is sent to the judgment module, and the cost incurred when the judgment result is sent to the control module. The data flow between the control module and the measurement module is carried out through the communication process between the user state and the kernel. When the judgment module goes to the TBDB to get or write the trusted reference value, the overhead increases by two environment switches, one shared memory registration, and one shared memory destruction. When performing  $n$  measured data transmissions, the entire process must go through two environmental switches, one shared memory registration, one shared memory destruction, four times intermodule communication, and  $2 * n$  times of communication between the kernel state and the user mode. As most of the measurements were performed in the REE, no world switching is required when handling external interrupts to avoid “starvation,” which reduces the impact on system performance.

It is, therefore, crucial to reduce the computational cost. Therefore, each environment switch must go through a complex process of state saving and loading, which is more costly in terms of performance than kernel-user interaction. Comparing the proposed architecture with the raw integrity measurement architecture, Table 5 shows that this mechanism greatly reduces the number of environment switches and optimizes performance.

**8.2.2. Performance Optimization Evaluation.** To verify the effectiveness of the architecture proposed in this paper, we used the Lmbench tool to test two systems with and without the security mechanism. The experiment was performed on a machine using a CPU of Intel Core (TM) i7-6500. It has two cores with 2.5GHZ and 4G memory. It uses OP-TEE to simulate ARM TrustZone and uses Linux as the operating system. The Linux kernel version is 4.15.0-142-generic. The

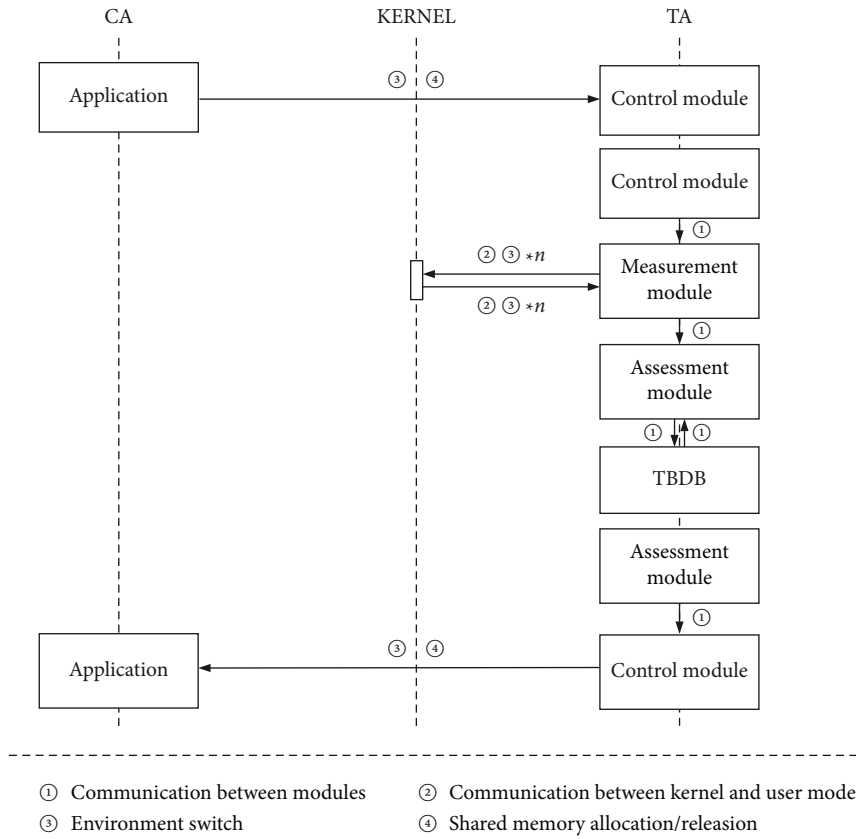


FIGURE 9: Raw integrity measurement architecture's additional overhead.

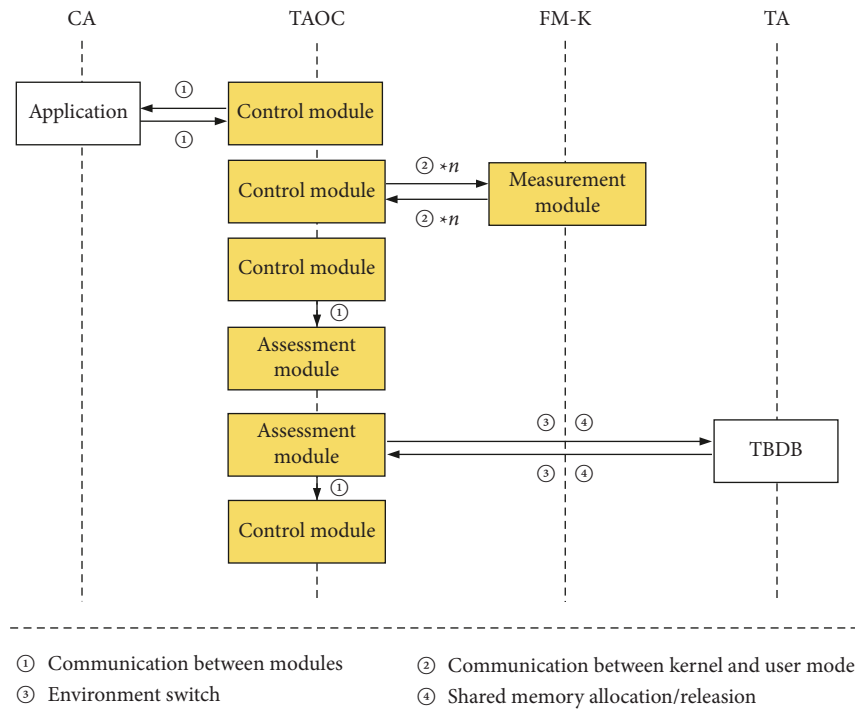


FIGURE 10: Extra cost of this design.

TABLE 5: Performance comparison of this architecture with raw integrity measurement architecture.

Operations	Raw architecture	Our architecture
Communication between modules	5	4
Communication between the kernel and the user mode	2	$2 * n^1$
Environment switch	$2 * (n^1 + 1)$	2
Shared memory allocation/release	2	2

<sup>1</sup> $n$ : number of times the measurement data are transferred.

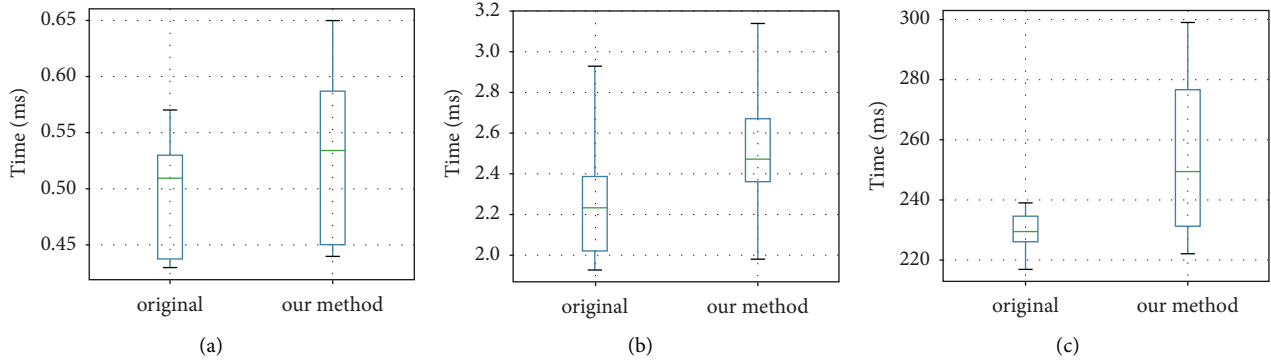


FIGURE 11: Testing normal operations. (a) Null-call operation latency. (b) Open-close operation latency. (c) Fork-proc operation latency.

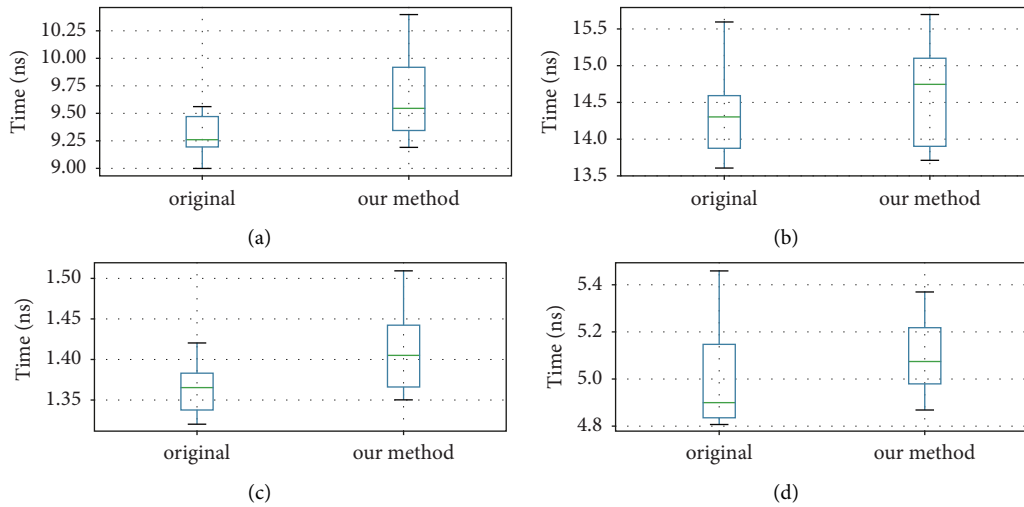


FIGURE 12: Testing basic integer operations. (a) Intgr-div operation latency. (b) Int64-mod operation latency. (c) Double-mul operation latency. (d) Double-div operation latency.

tests were divided into four categories: normal operations, basic integer operations, file system latencies, and memory latencies. Box plots were used to describe the experimental results. The test results are presented in the article as follows.

Normal operations included the *null-call*, *open-close*, and *fork-proc*. The *null-call* is a simple system call, and *open-close* is the operation of opening and then immediately closing the document. The *fork-proc* is the operation that allows one to exit immediately after forking the process. As can be seen in Figure 11, the impact of the security mechanism on the *null-call* operation was negligible, and the impact on the *open-close* operation was approximately 0.23 ms. Our architecture has some impact on the *fork-proc* operation. The impact was

about 20 ms, but the maximum delay was less than 80 ms. The basic integer operations include *intgr-div*, *int64-mod*, *double-mul*, and *double-div* to test the impact of hash operations on the system. The results in Figure 12 show that the largest impact on the four operations is approximately 0.5 ns. As can be seen in Figure 13, the file system latencies have two operations: *10k-create* and *file-delete*. Here, *10k-create* represents the time taken to create a 10k file, while *file-delete* represents the time taken to delete a 10k file. The delay of *10k-create* was increased by roughly 75 ms, and the impact of *file-delete* was about 20 ms. The operations in Figure 14 that affect memory are *main-mem* and *rand-mem*, which represent the latency of memory continuous access and

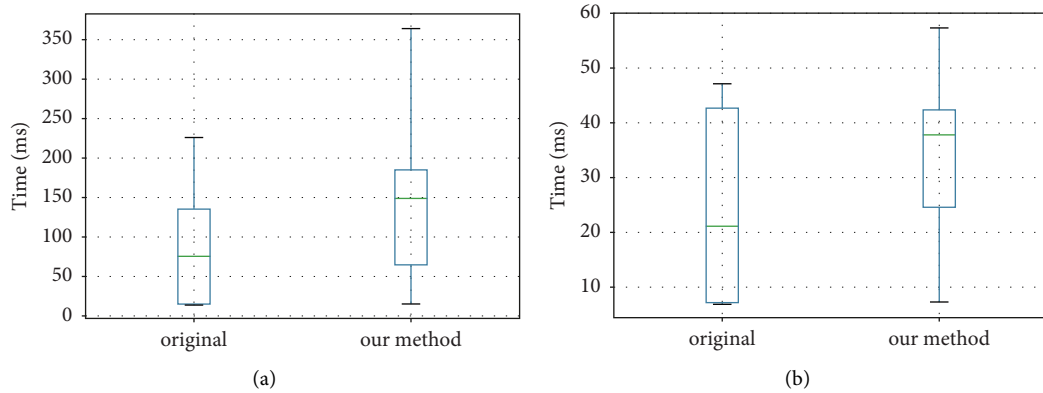


FIGURE 13: Testing file system latencies. (a) 10k-create operation latency. (b) File-delete operation latency.

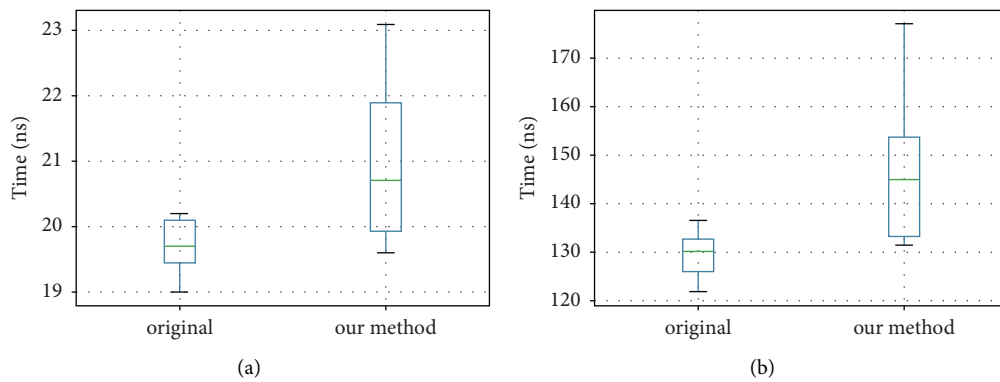


FIGURE 14: Testing memory latencies. (a) Main-mem operation latency. (b) Rand-mem operation latency.

random access operations. The delay of memory continuous access operations was approximately  $1.1 \text{ ns}$ . Compared with the system without a security mechanism, our system reduced the speed of memory random access operations by 11.54%. In summary, the impact of our method on the performance of the host system was very limited.

Moreover, the security mechanism is always on after the system is powered on, and the measurement data will be read and written frequently in memory. In order to evaluate the impact on the system, we measured the change in memory throughput in REE with or without a security mechanism. We performed 100 write and read memory operations under the memory space of 100 MB to analyze the average throughput. As shown in Table 6, the average write throughput is 3.49 MB/s when there is no security mechanism and 3.31 MB/s when there is a security mechanism. The throughput of write operations is reduced by 5.16%. The average throughput of read operations is 67.34 MB/s when there is no security mechanism, and it is 65.48 MB/s when there is a security mechanism. The throughput of read operations decreased by 2.76%. It indicates that the overhead of the normal computation system's memory is acceptable.

TABLE 6: Throughput testing with or without security mechanism.

Operations	Without security mechanism	With security mechanism
Write	3.49 MB/s	3.31 MB/s
Read	67.34 MB/s	65.48 MB/s

## 9. Conclusion

In the face of attacks on smart devices, technologies such as passive defense firewalls and traffic-based intrusion detection are no longer sufficient. Although the existing TEE technology can ensure system security to some extent, there are still some problems, such as the larger attack surface due to the large TCB, the insecure API of the TEE service, and the high-performance cost. In view of this situation, we developed a trusted computing active measurement architecture based on TrustZone. We reduced the size of TCB by removing some functional modules in TEE. We developed an active measurement mechanism to protect the internal security of the functional modules and the security of the target applications. It implemented active defense and defended against security vulnerabilities in the API interface



to the TEE service. By deploying measurement work on the REE, the world switches are reduced and the performance overhead is reduced. To improve the availability and efficiency of the system, a fallback mechanism is established as a response mechanism to attack. Experimental results have shown that our method can actively detect and respond to attacks. In addition, through performance testing and analysis, we have shown that our method can reduce performance overhead and reduce the impact of security mechanisms on the system. Currently, dynamic measurement only includes code segments and data segments. In the future, we will investigate the measurement of more dynamic structures, e.g., in the stack.

## Data Availability

The source code used to support the findings of this study can be made available from the corresponding author upon request.

## Conflicts of Interest

The authors declare that they have no conflicts of interest.

## Acknowledgments

This work was supported in part by the National Key R&D Program of China (Key Technologies and Applications of Security and Trusted Industrial Control System, grant no. 2020YFB2009500) and Beijing Municipal Natural Science Foundation (grant no. L192020).

## References

- [1] L. Zhou, C. Su, Y. Wen, W. Li, and Z. Gong, "Towards practical white-box lightweight block cipher implementations for iots," *Future Generation Computer Systems*, vol. 86, pp. 507–514, 2018.
- [2] X. Zhai, K. Appiah, S. Ehsan et al., "A method for detecting abnormal program behavior on embedded devices," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 8, pp. 1692–1704, 2015.
- [3] S. Iskhakov, A. Shelupanov, and A. Mitsel, "Internet of things: security of embedded devices," in *Proceedings of the 2018 3rd Russian-pacific conference on computer technology and applications (RPC)*, Vladivostok, Russia, August 2018.
- [4] H. Amrouch, P. Krishnamurthy, N. Patel, J. Henkel, R. Karri, and F. Khorrami, "Special session: emerging (un-)reliability based security threats and mitigations for embedded systems," in *Proceedings of the 2017 International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES)*, Seoul, Korea (South), October 2017.
- [5] C. M. VanYe, B. E. Li, A. T. Koch, B. E. Li, and M. N. Luu, "Trust and security of embedded smart devices in advanced logistics systems," in *Proceedings of the 2021 Systems and Information Engineering Design Symposium (SIEDS)*, Charlottesville, VA, USA, April 2021.
- [6] X. Lin, "The attack principle and defense of rootkit virus," *Computers & Security*, vol. 5, no. 4, 2009.
- [7] M. Botacin, M. Zanata, and A. Grégio, "The self modifying code (smc)-aware processor (sap): a security look on architectural impact and support," *Journal of Computer Virology and Hacking Techniques*, vol. 16, no. 3, pp. 185–196, 2020.
- [8] R. Farley and X. Wang, "CodeXt: automatic extraction of obfuscated attack code from memory dump," *Lecture Notes in Computer Science*, vol. 10, pp. 502–514, 2014.
- [9] S. Blazy, V. Laporte, and D. Pichardie, "Verified abstract interpretation techniques for disassembling low-level self-modifying code," *Journal of Automated Reasoning*, vol. 56, no. 3, pp. 283–308, Jan 2016.
- [10] AUDIT MY PC, "Linux kernel mprotect() function memory permission bypass," 2010, <https://www.auditmypc.com/network-security-4282006.asp> [Online]. Available:
- [11] M. Gentilal, P. Martins, and L. Sousa, "Trustzone-backed bitcoin wallet," in *Proceedings of the Fourth Workshop on Cryptography and Security in Computing Systems. ACM*, Stockholm, Sweden, January 2017.
- [12] Y. Fan, S. Liu, G. Tan, and F. Qiao, "Fine-grained access control based on trusted execution environment," *Future Generation Computer Systems*, vol. 109, pp. 551–561, 2020.
- [13] M. A. Bouazzouni, E. Conchon, and F. Peyrard, "Trusted mobile computing: an overview of existing solutions," *Future Generation Computer Systems*, vol. 80, pp. 596–612, 2018.
- [14] Global Platform, "TEE System architecture version 1.2," 2018, <https://globalplatform.org/specs-library/?filter-committee=tee> [Online]. Available.
- [15] X. Jia, Y. He, X. Wu, and H. Sun, "Performing trusted computing actively using isolated security processor," in *Proceedings of the 1st Workshop on Security-Oriented Designs of Computer Architectures and Processors. ACM*, Toronto, Canada, January 2018.
- [16] Arm, "Trustzone for armv8-a," 2020, <https://developer.arm.com/documentation/102418/0100/TrustZone-in-the-processor?lang=en> [Online]. Available.
- [17] J. Tian, H. Sun, X. Wu, and X. Chen, "A fine-grained trusted monitoring measurement method based on security-first architecture," *Journal of Cyber Security*, vol. 4, no. 5, 2019.
- [18] M. Dorjmyagmar, M. C. Kim, and H. Kim, "Security analysis of samsung knox," in *Proceedings of the 2017 19th International Conference on Advanced Communication Technology (ICACT)*, February 2017.
- [19] S. Dong, Y. Xiong, W. Huang, and L. Ma, "Kims: kernel integrity measuring system based on trustzone," in *Proceedings of the 2020 6th International Conference on Big Data Computing and Communications (BIGCOM)*, July 2020.
- [20] P. Dong, Y. Ding, Z. Jiang, C. Huang, and G. Fan, "Design and implementation of tpm/tcm with active trust based on tee," *Journal of Software*, vol. 31, no. 5, pp. 1392–1405, 2020.
- [21] Y. Qin, J. Liu, S. Zhao, D. Feng, and W. Feng, "Ripte: runtime integrity protection based on trusted execution for iot device," *Security and Communication Networks*, vol. 2020, pp. 1–14, Article ID 8957641, 2020.
- [22] H. Jiang, R. Chang, L. Ren, and X. Xiao, *An Effective Authentication for Client Application Using ARM Trustzone*, Springer International Publishing, Midtown Manhattan, NY, 2017.
- [23] Q. Zhang, J. Qiao, and Q. Meng, "Build a trusted storage system on a mobile phone," *IET Information Security*, vol. 13, no. 2, pp. 157–166, Mar 2019.
- [24] C. Brook, "Google removes rooting trojan dvmmap from play store," 2017, <https://threatpost.com/google%20removes-rooting-trojan-from-play-store/126111/>.

- [25] J. Amacher and V. Schiavoni, *On the Performance of ARM TrustZone* Springer International Publishing, Midtown Manhattan, NY, 2019.
- [26] Z. Ling, H. Yan, X. Shao et al., "Secure boot, trusted boot and remote attestation for arm trustzone-based iot nodes," *Journal of Systems Architecture*, vol. 119, Article ID 102240, Oct 2021.
- [27] S. D. Yalaw, P. Mendonca, G. Q. Maguire, S. Haridi, and M. Correia, "TruApp: A trustzone-based authenticity detection service for mobile apps," in *Proceedings of the 2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, October 2017.
- [28] S. Hu, Q. A. Chen, J. Joung, C. Carlak, and Y. Feng, "Cvshield: guarding sensor data in connected vehicle with trusted execution environment," in *Proceedings of the Second ACM Workshop on Automotive and Aerial Vehicle Security*, LA, New Orleans, USA, April 2020.
- [29] H. Sun, K. Sun, Y. Wang, J. Jing, and H. Wang, "Trustice: hardware-assisted isolated computing environments on mobile devices," in *Proceedings of the 2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015.
- [30] S. Pinto and N. Santos, "Demystifying arm trustzone: a comprehensive survey," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–36, 2019.
- [31] T. Hardjono and N. Smith, "Towards an attestation architecture for blockchain networks," *World Wide Web*, vol. 24, no. 5, pp. 1587–1615, 2021.
- [32] F. Raynal, "Overview of intel sgx," 2020, <https://blog.quarkslab.com/overview-of-intel-sgx-part-1-sgx-internals.html> [Online]. Available:
- [33] Y. Cheng, Q. Wu, W. Chen, and B. Wang, "Distributed shielded execution for transmissible cyber threats analysis," *Journal of Parallel and Distributed Computing*, vol. 122, pp. 70–80, 2018.
- [34] M. Orenbach, P. Lifshits, M. Minkin, and M. Silberstein, "Eleos: exitless os services for sgx enclaves," in *Proceedings of the Twelfth European Conference on Computer Systems*, April 2017.
- [35] Y. Zhang, W. You, S. Jia, L. Liu, Z. Li, and W. Qian, "Enclavepost: a practical proof of storage-time in cloud via intel sgx," *Security and Communication Networks*, vol. 2022, pp. 1–16, 2022.
- [36] L. Guan, P. Liu, X. Xing, X. Ge, and X. Zhang, "Trustshadow: secure execution of unmodified applications with arm trustzone," in *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, New York, Niagara Falls, USA, April 2017.
- [37] H. Sun, K. Sun, Y. Wang, and J. Jing, "Trustotp: transforming smartphones into secure one-time password tokens," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, October 2015.
- [38] A. S. Salman and W. Du, *Securing Mobile Systems GPS and Camera Functions Using TrustZone Framework*, pp. 868–884, Springer International Publishing, Midtown Manhattan, NY, 2021.
- [39] W. Li, H. Li, H. Chen, and Y. Xia, "Adattester: secure online mobile advertisement attestation using trustzone," in *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, May 2015.
- [40] X. Ge, H. Vijayakumar, and T. Jaeger, "Sprobes: enforcing kernel code integrity on the trustzone architecture," *Computer Science*, vol. 25, no. 6, pp. 1793–1795, 2014.
- [41] 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, pp. 1–16, 2020.
- [42] Y. Zhou, B. Zhao, and Y. An, "A novel trusted software base for commercial android devices using secure tf card," *Security and Communication Networks*, vol. 2022, pp. 1–12, 2022.
- [43] Z. Liu and D. Feng, "Tpm-based dynamic integrity measurement architecture," *Journal of Electronics and Information Technology*, vol. 32, no. 4, pp. 875–879, Jun 2010.
- [44] H. Yang and M. Lee, "Demystifying arm trustzone tee client api using op-tee," in *Proceedings of the 9th International Conference on Smart Media and Applications*, Jeju, Republic of Korea, June 2020.
- [45] D. Zhang and S. You, "Iflask: isolate flask security system from dangerous execution environment by using arm trustzone," *Future Generation Computer Systems*, vol. 109, pp. 531–537, 2020.
- [46] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using arm trustzone to build a trusted language runtime for mobile applications," *ACM SIGPLAN Notices*, vol. 49, no. 4, pp. 67–80, 2014.
- [47] Y. Zhang, Y. Qin, D. Feng, B. Yang, and W. Wang, *An Efficient Trustzone-Based In-Application Isolation Schema for Mobile Authenticators*, Springer International Publishing, Midtown Manhattan, NY, 2018.
- [48] B. Yang, P. Dong, L. Zhang, and Y. Ding, "Research on performance optimization of secure application based on trustzone," *Computer Engineering & Science*, vol. 42, no. 12, pp. 2141–2150, 2020.