PAPER

Malicious Code Detection for Trusted Execution Environment Based on Paillier Homomorphic Encryption

Ziwang WANG[†], ***Member and*** **Yi ZHUANG**[†a)], ***Nonmember***

SUMMARY Currently, mobile terminals face serious security threats. A Trusted Execution Environment (TEE) which can provide an isolated execution environment for sensitive workloads, is seen as a trusted relay for providing security services for any mobile application. However, mobile TEE's architecture design and implementation strategy are not unbreakable at present. The existing researches lack of detect mechanisms for attack behaviour and malicious software. This paper proposes a Malicious code Detection scheme for Trusted Execution Environment based on Homomorphic Encryption (HE-TEEMD), which is a novel detection mechanism for data and code in the trusted execution environment. HE-TEEMD uses the Paillier additive homomorphic algorithm to implement the signature matching and transmits the ciphertext information generated in the TEE to the normal world for detection by the homomorphism and randomness of the homomorphic encryption ciphertext. An experiment and security analysis proves that our scheme can achieve malicious code detection in the secure world with minimal cost. Furthermore, evaluation parameters are introduced to address the known plaintext attack problem of privileged users.

key words: *trusted execution environment, malicious code detection, mobile security*

1. Introduction

The Trusted Execution Environment (TEE), whose hardware-assisted security architecture provides greater security, integrity, and confidentiality guarantees, is widely used to provide an isolated execution environment for sensitive workloads. Under the guidance of the Global Platform standard [1] which is a secure area of the main processor, TEE implements rich security services like sensitive information storage, encryption and decryption, and security authentication based on various applications deployed in the secure world. Generally, the main idea of TEE is to minimize the Trusted Computing Base (TCB) and reduce the attack surface. Typical examples of TEE are ARM TrustZone technology [2] for mobile platforms and Intel Software Guard Extensions (SGX) technology [3] for desktop platforms.

ARM TrustZone is viewed as a de-facto standard for implementing a TEE on mobile devices, and also is a hardware security architecture for building an isolated execution environment in ARMv6 and all subsequent mobile processor chips [4], [5]. Thanks to the fact that TrustZone technology has been applied to most ARM mobile processor chips since 2001, more than 70 percent of mobile phones worldwide have integrated TrustZone processor chips [6]. With the development of the Internet of Things (IoT), the TrustZone module was also embedded in the new ARM Cortex-M series processor in 2015 [7]. Different from a common operating system running in the normal world, the TrustZone is intended to run a small and reduced kernel which is isolated from the rich OS, a.k.a. The secure world. TrustZone's powerful isolation capabilities and security sensitivity make it become an interesting topic for security researchers and mobile security practitioners. However, several vulnerabilities have been discovered, which could compromise the security of the TEE in TrustZone [8], [9].

The attack surface of TEE software mainly lies in the security vulnerabilities of the interface linking to the normal world and the trusted applications running in the secure world. Unlike trusted roots [10], TEE needs to provide rich trusted computing services for the normal world such as dynamic installation, dynamic updates, and interactions. On the one hand, service integration requires the creation of communication channels between the two worlds to share data, so it must accept inputs made by direct transfer or memory sharing from non-secure worlds and untrusted software. On the other hand, the increase of TCB also inevitably result in vulnerabilities that can be utilized by attackers. Fuzzing is an important way of implementing software attacks, which finds a vulnerability in the TEE by triggering an internal abnormal state or behavior of the secure world in the Client API and the TEE driver. In more cases, attackers implement code injection attacks based on the known vulnerabilities or problematic APIs. Hence, to address the serious risk of software attack for the TEE, it is necessary to study the malicious code detection mechanism.

There are two implementation strategies for providing malicious code detection services for the secure world. 1) Deploying detection programs in the secure world. However, this strategy is almost impractical for the following three reasons. First, a cropped microkernel running in the secure world cannot provide a runtime environment for the detection program. Second, implementing a complex detection program in TEE needs to extend the secure OS kernel to satisfy the detection program, but this operation can not only increase the TEE's attack surface but also affect the efficiency of the TrustZone. Finally, the update process of the virus database is inevitably complex and unbearable. 2) Deploying detection programs in the normal world. However, using

Manuscript received April 5, 2019.
Manuscript revised August 13, 2019.
Manuscript publicized September 20, 2019.

[†]The authors are with Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China.
a) E-mail: zy16@nuaa.edu.cn

DOI: 10.1587/transcom.2019EBP3098

a low-security environment to provide detection services for a high-security environment will obviously bring a significant security risk. Therefore, we present a Malicious code Detection scheme based on Paillier Homomorphic Encryption in the Trusted Execution Environment (HE-TEEMD) which can effectively solve the facing problems of strategy two.

The Paillier homomorphic cryptosystem and first homomorphic algorithm support any times of additions based on the problem of computing n-th residue classes that determine it has higher encryption and decryption efficiency. HE-TEEMD transfers the data to be detected from the secure world to the normal world by the homomorphism and randomness of the Paillier homomorphic encryption ciphertext. A ciphertext malicious code detection scheme has been established. More specifically, HE-TEEMD encrypts the data to be detected in the secure world with Paillier encryption algorithm and transmits the ciphertext data to the normal world. A detector application running in the normal world discover the malicious code from the ciphertext data.

We have envisaged two application scenarios for this proposal. First, malicious code analysis is implemented, which means that the malicious code fragments contained in the data encapsulation or shared memory are detected by signature matching. Second, in order to identify virus files and applications, we have two different strategies to deal with a suspicious file. For quick detection, we can distinguish whether the file to be detected in the secure world is a virus file by matching the MD5 hash. For deep detection, we can search byte by byte for malicious code fragments contained in the target file or program.

We believe that this is the first implementation of malicious code detection against TrustZone. Our main contributions in this paper are summarized below.

- First, a novel TEE malicious code detection scheme based on Paillier homomorphic encryption is proposed.
- Second, by introducing the evaluation variables, the competition of modifying and integrity verifying of the detector is solved.
- Third, the solution has been evaluated to measure the impact on the real-time performance of the system. Experimental results show that HE-TEEMD achieves a minimum of TCB and a small performance cost.

The remainder of the paper is organized as follows: Section 2 reviews relevant researches and explains the necessity of our work. Section 3 briefly describes ARM TrustZone and Paillier Homomorphic encryption. Section 4 discusses the threat model and assumptions of this work. A novel malicious code detection architecture, HE-TEEMD, is illustrated in Sect. 5. Section 6 details the implementation of the algorithm. We also present the validity of our algorithm and the evaluation of the performance in Sect. 7. Section 8 is the conclusion.

2. Related Work

Because the current mobile TEE's architecture design and implementation strategy are not unbreakable, a mechanism to detect the data and code in the trusted execution environment is necessary. With the wide application of TEE technology, the research on attacks against TEE has become a hotspot of current security research. The security threats of TEE on mobile platforms are mainly divided into two aspects: 1) Attacks on the TEE's security architecture or its implementation strategy. Many researchers have tried to achieve attacks on the TEE's architecture. For example, CLKSCREW [9] attack uses software-exposed frequency and voltage hardware regulators to extract keys, and this mechanism loads self-signed applications in commercial TEE to destroy TrustZone security. Armageddon [8] points out that TrustZone's cache activity can be detected from normal world, and researchers use Prime+Probe cache attack to distinguish whether a fake key is valid. 2) Attacks on the program running in the TEE [11], [12]. On the one hand, because the security of the TEE depends on the integrity of the trusted applications and the secure world OS kernel, any software error would cause the secure world collapse. On the other hand, due to the strong interactivity of TEE, the universal TEE architecture directly exposes the secure world interface to non-privileged users in the non-secure world [2], which means that any non-privileged user can attack the TEE. Additionally, a lot of attacks exploit vulnerable error codes to destroy TEE security [13], [14]. Multiple security vulnerabilities have been discovered in the TEE kernel of major vendors [15]–[19]. Dosomder et al. [16] found a security vulnerability in the Qsee Communicator program file that could allow an attacker to exploit the vulnerability with a specially crafted application. In [17], the authors proposed that if the attacker obtains the root permission of some Huawei mobile devices, it is possible to implement a denial of service (secure world OS crash) or implant and execute malicious code in the secure world OS by transmitting an abnormal address to the TEE. In the open-source TEE solution OP-TEE [18], [20] proposed a security vulnerability in the encrypted dynamic library file code that an attacker can be used to recover the private key.

However, the malicious code or data not only could be installed by exploiting vulnerabilities mentioned above, but also can trick users installing into the secure world directly. So, how to identify the suspicious code and data is one of the objectives of this study. Researchers have tried to build solutions for TEE in different ways, but existing implementations still rely on the non-secure world operating system to disinfect any input before passing it to the secure world [11]. In [11], Zhenyu Ning et al. proposed that damaged TEE can be detected by TEE's execution mode exceptions, such as sending sensitive memory pages, frequently calling security services, and keeping the secure world for a long time. Formalization of the API can also increase the security of the TEE [21]. Although Global Platform requires that

the TEE solution certified by its specification must pass the functional API specification certification testing, the API's security for user-installed trusted applications still cannot be guaranteed.

Recently, a few researchers have attempted to build malicious code detection services in the secure world directly. T2Droid [22] is a malware dynamic analysis scheme on Android-based mobile devices, which attempts to deploy the APK detection program into the TEE for a higher-privilege detector. T2Droid sharply increases the trusted computing base of the secure world, which consequently increases the risk of having security vulnerabilities. Similarly, TZ_KPM [23] implements kernel code integrity checking and malicious processes detection for the normal world by a series of complex mechanisms in TEE. Building a detection program in the secure world to provide malicious code detection services for the normal world (android) has a certain practical significance. However, it cannot be used to improve the security of the secure world itself. The reason is that the scheme itself is sure to reduce the security of the TEE greatly. To the extent of our knowledge, literature [11] is the only architecture present that declares to provides security detection services for the TEE. However, it can only determine whether the secure OS kernel is damaged based on some indirect data such as calling frequency anomalies but cannot directly read the original data of the secure world for malicious code analysis.

LOCAL, a mechanism which builds the detector in the secure world, is implemented in our research work. In fact, there is not any manufacturer or researcher to deploy malicious code detection programs in TEE to improve the security of TEE. On the one hand, it is not advisable due to those reason mentioned in the previous paragraph. On the other hand, for a long time, TEE has been considered to provide adequate security for low-value devices or low-resource things. However, with the popularity of devices with TEE and a growing number of attacks against TEE, malicious code detection mechanisms are indispensable.

To the best of our knowledge, HE-TEEMD is the only attempt to provide malicious code detection services to the secure world, with minimal TCB and performance cost. It should be pointed out that HE-TEEMD does not directly defend against those existing attacks mentioned above for TEE, it only works for inject attacks and a small number of known attacks, such as FUZZING attack and replay attack. Similarly to anti-virus software, the more significant role of HE-TEEMD is to discover the fact of attack that is already present in the TEE.

3. Background

This section provides background information about the main technologies of HE-TEEMD. Initially, an introduction to ARM TrustZone is given. Then follows a discussion on the supporting technology of the proposed mechanism, the Homomorphic Encryption algorithm and the Malicious code analysis technique. Furthermore, we exhibition the reason why static analysis techniques are more suitable for Trusted Execution Environments.

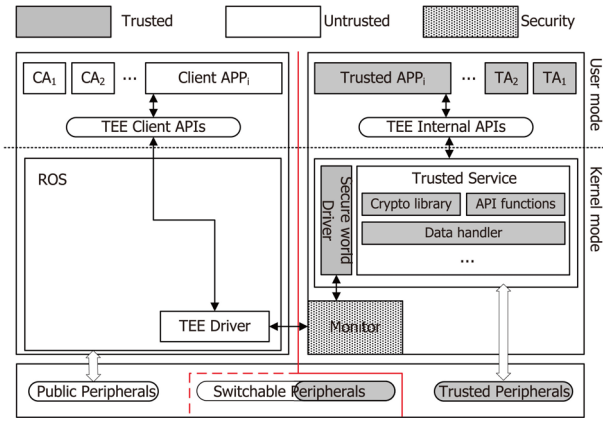


Fig. 1 The architecture of TrustZone.

3.1 ARM TrustZone

Being a hardware security architecture, ARM TrustZone virtualizes two completely isolated execution environments through hardware assistance [2], the secure world and the normal world. TrustZone virtualizes the CPU into two cores in a time-division manner. The underlying hardware resources (registers, physical memory, peripherals, etc.) are also dynamically divided into secure and non-secure. It switches between those two cores through monitor mode. In general, the normal world runs the Rich Operating System (ROS), such as Linux. The secure world runs a lightweight and trusted operating system called a TEE. This architecture is widely adopted by Android, iOS and other mobile operating systems.

The architecture of TrustZone is shown in Fig. 1. TrustZone controls the switching between the two worlds through hardware logic. When actually running, the user program first calls the TEE Client API, which enters the kernel-mode from the user mode through the system call, finds the relative TEE driver according to the corresponding parameters, and then triggers the runtime system in monitor through the SMC or interrupt. The runtime system saves the context of the normal world and changes the Non-Secure (NS) bit to switch to the secure world so that it can enter the TEE Trusted OS [4], [5].

The TrustZone Protection Controller (TZPC) technology allows for restricting the system devices to the secure world or the normal world, which dedicates one bit to each individual device. TZPC allows the secure world to have exclusive control of security-critical peripherals so that the secure world can explicitly configure and access peripherals. TZPC also supports dynamically assigns security-critical peripherals to one of two environments and prevents non-secure access.

3.2 Homomorphic Encryption

Homomorphic encryption [24] refers to a special type of encryption algorithm that allows performing a specific algebraic operation, generating an encrypted result which matches the result of performs the same operation on the plaintext. Homomorphic encryption allows the computing party to calculate ciphertext data without knowing the original plaintext. Simply put, the value of $x_1 \circledast x_2$ can be calculated directly in the case where only the ciphertext data Y_1 and Y_2 are known and the x_1, x_2 is unknown [25]. Homomorphic encryption technology has been widely used in the fields of trusted computing and cloud computing.

The Paillier cryptographic algorithm is a public-key cryptosystem with semantic security, additive homomorphism and ciphertext randomness. Its theoretical security is built on the Computational Composite Residuosity Assumption (CCRA). Semantic security means that for a given plaintext m_1 and m_2, and one of the plaintext correspondings to ciphertext C, there is no polynomial algorithm to prove to which plaintext the ciphertext C corresponds. Additive homomorphism means that given any unclear x and y, $E_{pk}(x; r_1) \cdot E_{pk}(y; r_2) = E_{pk}(x + y; r_1 r_2)$ is satisfied. Ciphertext randomness means that the same plaintext does not generate the same ciphertext, so the attacker cannot crack and obtain information through statistical analysis of ciphertext.

3.3 Malicious Code Analysis

There are many ways to detect malicious code on a mobile platform. These methods can be broadly divided into two types: static and dynamic analysis [26]. The static analysis method detects whether the application or instruction is malicious by examining the code or metadata. Virus signature scanning technology is the most common static analysis technology. Firstly, analyze the malicious code to extract the signature different from other programs, and then keep the signature into the signature database and use this signature as the code flag to scan the specific virus. Static analysis has low requirements for the running environment, thus can save the system resources of analysis and scanning. Dynamic analysis detects malicious programs according to the specific behavior and execution results of the program during execution. It can identify unknown types of attacks but will cost a lot.

However, dynamic analysis is not suitable for the trusted execution environment, as shown in Table 1. On the one hand, as a more complex and energy-intensive detection technology, dynamic malicious code analysis technique usually has a huge amount of code, which could significantly increase the TCB of the secure world. Then a time-consuming detect program cannot provide runtime services for the secure world. On the other hand, the feature of TEE limits the dynamic analysis technique as an optional option. First, programs in TEE typically do not produce complex behavioral models, simple behavioral samples cannot support the implementation of dynamic detection techniques. Second, the OS in TEE is usually difficult to update for known vulnerabilities, the static analysis method is more efficient for a known attacks activity. Third and most importantly, mobile platforms generally are low-value, we need to study how to survive from mass homogenization attacks (such as Botnet) with minimal cost, instead of studying more accurate detection models.

Table 1 The result of performance comparison.

ANALYSIS METHOD	POSITIVE	NEGATIVE
Dynamic	Unknown attacks	Huge TCB Lack of training samples Time Consuming
Static	More effective Small TCB	Unknown attacks

3.4 OP-TEE

OP-TEE is an open-source TEE operating system running on ARM TrustZone, which follows the Global Platform's TEE System Architecture Specification [20]. In the OP-TEE architecture, an application is divided into two parts. One is the Client Application running in the ROS, and the other is the Trusted Application running in the TEE. This means that security-sensitive services are extracted to a separate security environment to run.

4. Threat Model and Assumptions

This section describes the threat model and assumptions pertaining to the HE-TEEMD architecture.

4.1 Assumptions

As mentioned in prior sections, HE-TEEMD is built on the TrustZone hardware-assisted universal TEE architecture to provide a malicious code detection implementation with minimal TCB. We first explain the main idea of our design and then explain the security mechanism.

4.2 Threat Model and Assumptions

We assume that the mobile device is equipped with an ARM TrustZone.

First, we consider that there is a lightweight, trusted OS running in the secure world, and a fully compromised rich OS running in the normal world — a malicious user with root privileges exists in the normal world.

Second, assume that the original components running in the TEE are entirely trusted, including all components in the secure world side in our scheme. The application installed by the user is considered suspicious, this means that even if an application is successfully installed into the secure world, we still need to detect it.

Third, the attacker has complete control over all the software in ROS.

Fourth, an attacker with ROOT privileges can create a malicious process that continuously sends requests with well-designed parameters to discover the vulnerabilities of the secure world kernel and applications through the API or shared memory calls provided by the trusted application.

Fifth, an attacker may listen to the crossing-data between the secure world and the non-secure world to steal sensitive information.

Finally, we assume that the attacker did not implement physical access to the mobile device.

4.3 Threat Model

Threat models for HE-TEEMD involve two main aspects. The first one is any attacks to the component which runs in the normal world. The second involves attacks from the evil data transmitted to the secure world. For the first aspect, a privileged user can destroy the availability and integrity of any client application running in the normal world, but the confidentiality can be protected by encrypting and confusing. For the second aspect, two kinds of data are considered which pose the main threat to the security of the secure world: one is the crossing-data between the two worlds, including the payload contained in the request message from the normal world to the secure world, or the memory address of a file to be invoked by the secure world. Another one is the source code or data of trusted applications installed by the users in the secure world.

To mitigate the FUZZING or replay attacks, any crossing-data from the normal world to the secure world should perform a real-time detection process. To detect a file or program, which is commonly more than 10 kb, a better choice is to deal with it as a free time task.

5. Architecture and Design

5.1 Our Design

The security of TEE depends not only on small TCB and strong isolation but also on the security of the code running in the secure OS. How to achieve malicious code detection with minimal TCB growth cost is the focus of HE-TEEMD design.

In a general TEE architecture based on TrustZone, the user program *CAi* sends the security service request and data through the API encapsulation to the TEE driver. TEE driver achieves the switch to the secure world and transfers data to the trusted OS in the secure world. The trusted OS loads the corresponding security service program *TAi* according to the *UUID* in the request and transmits the data, and then *TAi* performs security services and returns the results. Its standard transmission path is shown in Fig. 2(a).

In the above process, the trusted OS does not perform malicious code detection on the incoming data of the trusted application, or only performs simple verification, which could not guarantee the security of the data. Unlike the general architecture, HE-TEEMD detects malicious code by encrypting plaintext information in the secure world and outsourcing it to a detector program running in the normal world, only the data that the detector determines to be benign will be sent to the trusted application for execution. Its transmission path is changed as shown in Fig. 2(b).

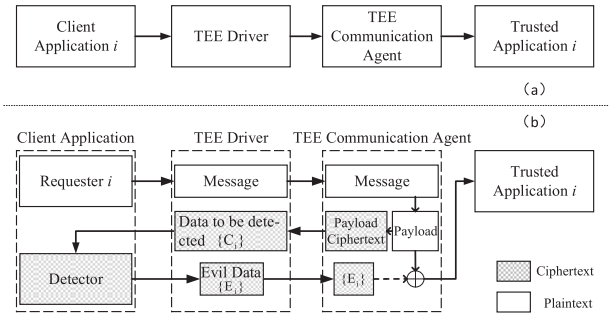


Fig. 2 The path of data transmission.

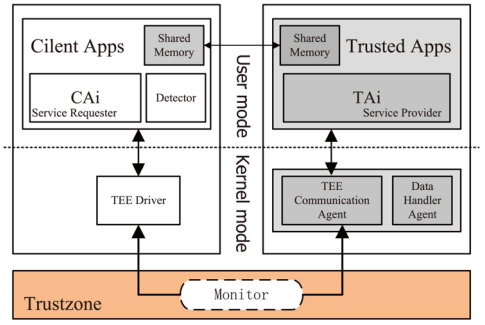


Fig. 3 The architecture of HE-TEEMD.

Figure 3 shows the architecture of HE-TEEMD in which ROS and trusted OS are running in the normal and secure world respectively. The communication between the two worlds is realized through shared memory. In HE-TEEMD, we modify the code of the TEE Communication Agent so that it can obtain and encrypt the data to be detected, and then send ciphertext to the detector running in the normal world. The detector is running in the non-secure environment with no guarantee of confidentiality and availability. And we use homomorphic encryption and evaluation parameters to establish a trusted communication link between the two worlds.

For comparison, we built a malicious code detection program running in the secure world, called LOCAL. The result of the comparative analysis between the LOCAL and the HE-TEEMD is illustrated in Table 2. It was clear that LOCAL is unfeasible. First, LOCAL will cost huge storage and update cost. On the one hand, the virus signature database can only be stored in secure memory to prevent attackers from editing it. On the other hand, writing a virus database into the secure world requires complex authentication protocols, and the virus database is a component that requires frequent updates in a detection system. Second, LOCAL will cause the TCB swift growth, the program of the

Table 2 The comparison of LOCAL and HE-TEEMD.

SCHEME	HASH CHECK	SIGNATURE MATCHING	SOLUTIONS
LOCAL	TCB Growth Storage Cost Update Cost	TCB Growth Storage Cost Update Cost Time Consuming	Expand Hardware
HE-TEEMD	Confidentiality	Confidentiality Time Consuming	Homomorphic Encryption

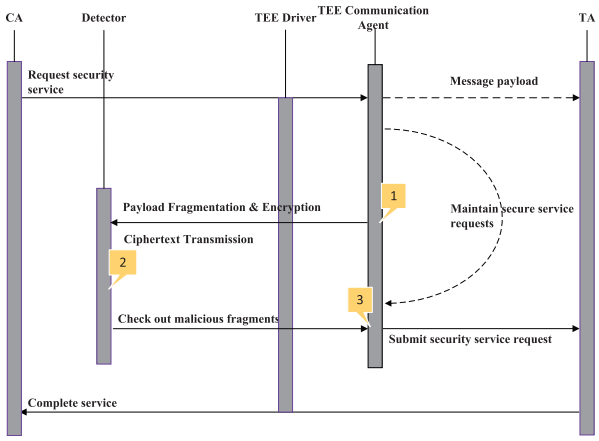


Fig. 4 The execution flow chart of HE-TEEMD.

detector will also cause some new vulnerabilities. To solve these problems, this paper proposed a complete malicious code detection architecture based on Paillier homomorphic encryption with minimal TCB and performance cost.

HE-TEEMD only adds a detector component to the user mode of the normal world without needing to customize and add any complex security components to the secure OS, which means that the proposed mechanism can be used in any commercial TEE solution. The detector is located in the user mode of the normal world. Its availability and confidentiality can not be guaranteed, so the data out of the secure world must be ciphertext, and the ciphertext calculation requires ciphertext with operability. Since it is not transparent to TEE driver in the transmission process of ciphertext, which means TEE driver can have both plaintext and ciphertext information of crossing-data, the ciphertext is required to have randomness.

HE-TEEMD implements the operability and randomness of ciphertext through the homomorphic encryption algorithm. With the HE-TEEMD, the suspicious data in the secure world can be encrypted and transmit to the normal world for a malicious code detection services.

Assuming there is a communication data m, its execution flow is shown in Fig. 4. The encryption calculation *Enc* is implemented in Fig. 4 ①and the homomorphism calculation *Cal* for a ciphertext at ②, a ciphertext fragment whether contains any malicious code is determined according to the calculation result. In ③, the decryption calculation *Dec* is implemented, the ciphertext segment of the malicious code is decrypted, and then whether the data m can be transmitted to the trusted application for execution can be judged according to the plaintext.

HE-TEEMD can detect malicious code for the following three types of data: 1) Payload in the interaction data of two world. No matter what the data structure TEE Driver transmits to the secure world, its payload is usually a collection of parameter variables or string fragments. This kind of data generally less than 1 KB, which could produce a small delay in real-time detection. 2) A file or its memory address. To detect a file can either perform a rough detection by Hash or a deep detection byte-by-byte, the latter is not suitable as a real-time service. User programs in the secure world are typically single-function, with the size of the code usually not more than 1 MB. It is feasible to scan malicious code signatures for trusted applications code under the architecture of HE-TEEMD. And 3) Internal function output, especially the decryption component. If a trusted application gets a ciphertext data from the normal world, the secure OS should intercept the output of the decryption function for detection. In the static detection process, the detection algorithms of the above three kinds of data are the same. However, in the actual application scenario, data type 1 as the main entrance for malicious code to enter the secure world, not only requires an effective detection mechanism but also needs to generate a brief delay when processing data 1 to provide real-time detection service. Therefore, the interaction data is used as the detection load in the subsequent verification and implementation process.

5.2 Component

HE-TEEMD mainly consists of five components: the client applications, the TEE driver, the TEE communication agent, the detector, and the trusted applications.

The Client Applications. The client applications is a program running in the normal world and are the initiator of the security service request. In fact, the detector is also a client application. In this paper, the client application refers specifically to the initiator of the security service request.

The TEE Driver. The TEE Driver runs in the kernel of the normal world for handling requests and corresponding commands when switching between the two worlds of TrustZone. It contains the crossing-data of this two world, which means that the crossing-data of the two worlds under the general TrustZone architecture is not transparent to the TEE Driver. Each TEE has a dedicated, non-trusted TEE Driver, each of which uses its own unique calling convention and data structure for data transferring. In this article, we simulate an attacker capability model with root privileges through a fully compromised TEE Driver. It is means that the attacker can monitor and modify the crossing-data between the two worlds and can prevent the detector from serving the secure world.

The TEE Communication Agent. The TEE communication agent is a module running in the secure world. HE-TEEMD requires some lightweight modifications to the

traditional TEE communication agent component, the new functions of the TEE communication agent is mainly used to intercept the data to be detected, such as the corrosing-data from the normal world to the secure world, the code fragment of the trusted applications program, and the output of TEE internal function. The trusted kernel guarantees that the data entering the trusted application will pass through the TEE communication agent module. The plaintext information to be detected intercepted by the TEE communication agent will be homomorphically encrypted and sent to the detector in the normal world to obtain the detection result.

The Detector. The detector is the main component of the HE-TEEMD running in the normal world, performing static malicious code analysis on the input data. Software protection technology is used to ensure its integrity.

The Trusted Applications. The trusted applications is a program running in the secure world and is a provider and responder of security services.

To sum up, HE-TEEMD only needs to modify the TEE Communication Agent component to intercept the data to be detected, and then send the cipertext back to the normal world. A normal Detector installed to the normal world, running a variety of static analysis programs, such as signature matching, string statistics, similarity comparison and so on. In the following chapters, the Homomorphic Encryption Algorithm and the Evaluation Variables are used to ensure the availability and security of HE-TEEMD. We prove that Paillier homomorphic encryption algorithm can achieve signature matching in ciphertext domain by using mathematical proof (chapter 6.1). Briefly speaking, if the character strings to be detected is format into a sequence of integers $\{i\}$, there must be a negative integer sequence $\{-i\}$ to make the ciphertext product of the two equal 0. And then, we evaluated the performance of the detection algorithm (chapter 7), the result shows that HE-TEEMD can realize a certain strength signature matching within a few milliseconds.

6. Protection Mechanisms

Three mechanisms implemented to guarantee the availability and security of HE-TEEMD: the homomorphic encryption, the ciphertext detection algorithm, and the evaluation parameters. The homomorphic encryption and the ciphertext detection algorithm not only ensure that the operability, the confidentiality and the privacy of the data outsourcing to the normal world, but prevent attackers from performing statistics and analysis on ciphertext data. The availability of the detector running in the normal world is the most important attributes, HE-TEEMD defends the availability attack and the forgery attack to the detector by the evaluation variables. This section will describe them in detail.

6.1 The Paillier Homomorphic Encryption

In this paper, the Paillier homomorphic encryption technology is used to process the information to be detected into a ciphertext sequence. And the ciphertext is transmitted to the normal world for detection. It is required that the security should be guaranteed even if the TEE driver is not trusted. To achieve this security, ciphertext information must meet the operability and randomness. Homomorphic encryption is a good way to implement the required security attributes, as it allows a specific algebraic operation of ciphertext to obtain the same result as the operation of plaintext. HE-TEEMD can achieve similarity-based detection algorithms such as signature matching with the addition homomorphism of the paillier homomorphic encryption algorithm [27]. It does not need to restore the data to be detected into plaintext, which can satisfy both homomorphism and randomness.

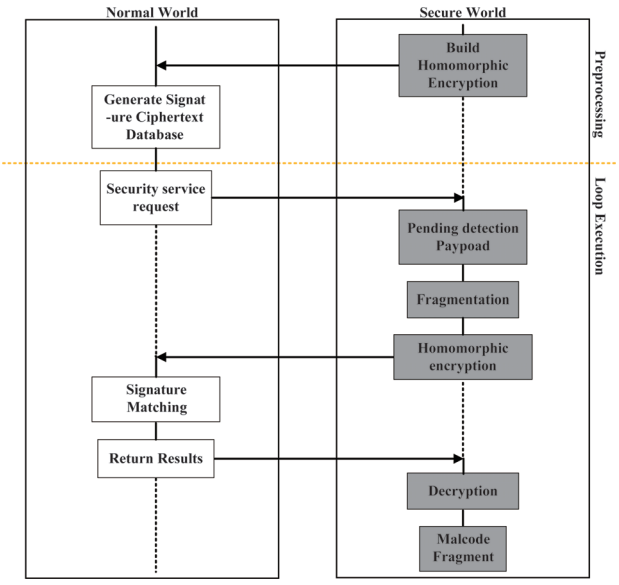


Fig. 5 The homomorphic encryption flow chart.

The process of implementing a complete homomorphic encryption algorithm by HE-TEEMD needs to be switched between the secure world and the normal world. As shown in Fig. 5, the signature ciphertext database is generated by the trusted service provider first, they encrypt their plaintext virus database into a ciphertext database by using the user public key and pushes it to the user local. The signature of the malicious code is divided into fixed-length fragments that represent each piece of plaintext information as a unique integer. Finally, it only needs to decrypt the product of the ciphertext and verify whether the decrypted result is 0 or not to determine whether the plaintext segment m matches the feature segment f. Further, we propose a matching check function, which can quickly determine whether the result of the homomorphism calculation function is 0 without decrypting the clear text result. The specific steps are as follows:

6.1.1 Build Paillier Homomorphic Encryption

Two large random primes p and q are generated to meet $\gcd(p \cdot q, (p-1) \cdot (q-1)) = 1$. The product $n = p \cdot q$ is calculated and generate random number g ~~$(g \in \mathbb{Z}_{n^2}^*)$~~, set

$g \epsilon {Z_{n^2}}^*$ where ${Z_{n^2}}^*$ represents a set of integers in Z_{n^2} that are mutually prime with n^2. $\mu = \left(L\left(g^\lambda \bmod n^2\right)\right)^{-1} \bmod n$ is required to exist, where L is a function of μ, denoted as $L(\mu) = \frac{\mu-1}{n}$. $\langle n, g\rangle$ is the user's public key and $\langle \lambda, \mu\rangle$ is the user's private key, where $\lambda = \mathrm{lcm}(p-1, q-1)$.

In theory, a larger value of p and q means that the cryptographic algorithm will have higher security. The optimal size of p and q is 128 bits for Paillier, which gives a high-security level and minimal performance loss [28]. Therefore, in our experiment code, both p and q are set to 128 bits.

6.1.2 Paillier Encryption Function

Grouping payload M into a number of fixed-length messages, each of which can be represented as a unique integer $M = m_1 m_2 m_3 \cdots m_i$. Calculate the ciphertext using the encryption algorithm demonstrated in Eq. (1) [27], where r_i is a random number.

$$c_i = E(m_i) = g^{m_i} \cdot {r_i}^n \bmod n^2 \tag{1}$$

6.1.3 Paillier Homomorphic Calculation Function

The calculating method of verifying ciphertext information to meet the paillier addition homomorphism is shown in Eq. (2) [27]. Where r_i and r_j are random numbers.

$$\begin{aligned} E(m_i)\, E\left(f_j\right) &= c_i \cdot c_{f_j} = g^{m_i} \cdot {r_i}^n \cdot g^{f_j} \cdot {r_j}^n \bmod n^2 \\ &= g^{m_i + f_j} \cdot \left(r_i \cdot r_j\right)^n \bmod n \\ &= E\left(m_i + f_j\right) \end{aligned} \tag{2}$$

According to Eq. (2), we can get the corollary 1.

Corollary 1: if $m_i + f_j = 0$ is true, then $E(m_i)E(f_j) = E(0)$ is also true.

6.1.4 Paillier Decryption Function

Paillier decryption function is shown in Eq. (3) [27].

$$D(c) = \frac{L\left(c^\lambda \bmod n^2\right)}{L\left(g^\lambda \bmod n^2\right)} \bmod n \tag{3}$$

6.1.5 Matching Check Function

HE-TEEMD send the public key $\langle n, g\rangle$ to the service provider to generate the signature database, and the public key is unknown to the local normal world. Part of the private key will be sent to the detector in the normal world to judge weather the plaintext fragment m is matched with the signature fragment f according to the Eq. (4).

$$L\left(c^\lambda \bmod n^2\right) \bmod n = 0 \tag{4}$$

It can be known from Eq. (4) that all the data required to complete a matching operation in the normal world is $\langle n, \lambda\rangle$. According to Eqs. (1), (2) and (3), it is safe to disclose such information to the non-secure world.

6.2 The Ciphertext Detection Algorithm

As is mentioned above, when a and b are negative on the integer field, the product of their Paillier homomorphic ciphertext $E(a) * E(b)$ must be 0 on the integer field after decryption. This means that we only need to convert the data to be detected and its signature to a sequence of integers, it is possible to determine the result is matching based on whether or not the products plaintext of the two ciphertexts is 0. So when Eq. (4) is true, the result of matching must be true according to Eq. (3). In fact, Eqs. (3) and (4) time consumption is the same as a check function. Besides, through the isolation of hardware peripherals by TrustZone, a homomorphic encryption chip can serve both the secure world and the normal world. On the one hand, the encryption chips format the result of Eq. (3) to 0 or 1, and output them to the normal world. On the other hand output the original result of Eq. (3) to the secure world. This can reduce the time consumption of repeatedly computing the same data in both worlds. And most importantly, it eliminates the TCB growth that implements homomorphic encryption in the secure world. To achieve a minimized TCB, we implemented a performance evaluation by software-simulating switchable hardware peripherals in the experiment in Sect. 7.

The detector, as the executor of the homomorphic ciphertext matching algorithm, receives the data to be detected and implements the malicious code detection algorithm. The detector mainly achieves the following two functions: using Hash to identify virus files and matching malicious code signature fragments. The trusted service provider can further generate some simple ciphertexts of logical code, such as count the times of calling to the $LibTomCrypt$ dynamic library $Times_{LibTomCrypt}$ in a period of time.

Generally, being a component that needs to be updated frequently, the detector has a complex iterative check mechanism which will not only increases the risk of compromise but also can not guarantee its timeliness. Sileshi D. Y. et al. [22] proposed that the use of complex integrity verification protocols for detector running in the non-secure world would introduce the competition of modifying files multiple times. In order to implement a detector in the non-secure world, HE-TEEMD uses the homomorphism and randomness of the ciphertext to reduce the security requirements of the detector.

It has to be noted that there is only the typical static analysis function has been build in this paper, such as signature matching, similarity comparison, and call frequency statistics. The focus is to demonstrate the feasibility and security of the ciphertext detection mechanism based on Paillier homomorphic encryption.

6.3 The Evaluation Variables

Availability is another crucial requirement for the detector to provide a trusted service. The evaluation variable is used to ensure the availability of the detector. Briefly speaking, the correct detection result can only be output to the evaluation variables when the detector is available.

Since the communication data of the two worlds are transmitted by TEE driver, the TEE Driver may have both plaintext and ciphertext for the detected data in the detector. It is still able to predict the output of the detector through a known-plaintext attack in the case of an unknown encryption key, and forge the detector output.

Supposing a malicious user injects an attack code e into the data $M\,(m_1, m_2, m_3, e, m_4, ...)$ passed to trusted application, the information that can be stolen by the attacker through TEE driver includes the plaintext M, the ciphertext $C\,(M)$, and the ciphertext fragment $C\,(e)$ of the malicious code e that the detector feeds back to the TEE communication agent. An attacker with root privileges can prevent the secure world from obtaining the correct detection results by tampering or forging a communication process. HE-TEEMD addresses the above-mentioned attacks by adding evaluation variables, which means adding several strings $A\,(a_1, a_2, a_3, a_4...)$ with known detection results after the plaintext information M. Makes the malicious code that the secure world gets from $C\,(e)$ to $C\,(e) + C\,(A)$. Because of the randomness of homomorphic ciphertext, it is guaranteed that the attacker cannot distinguish the ciphertext generated by different plaintexts or even the ciphertexts generated by the same plaintext at different times. This means that an attacker cannot forge the detection result of malicious code e.

The secure world further determines whether the detector's detection result for the payload M is reliable by the accuracy of the detection result of the evaluation variables, which could avoid the competition for the integrity check of detectors running in the normal world.

We assume that an attacker can forge or tamper with the output of the detector, and pass the cryptographic verification successfully. However, an attacker cannot just tamper with the information they want to hide and retain the information they want to maintain. First, the attacker cannot get the correct detection result of the evaluation variables by constructing a fake detector program. Second, the attacker also cannot identify which part of the detection result is generated by the evaluation variables. This means that only true and available detector programs can output correct results, and abnormal detection results equal to an abnormal detector or ROS. Once the wrong or unreached detection results are found, some security operations should be performed in the normal world, such as system initialization.

7. Evaluation

We cross-compile an OP-TEE trusted execution environment for the QEMU platform in the Ubuntu 14.04.1 system. Ubuntu runs on a single-core 2.7 GHz, 4G memory hardware platform, on which a prototype of a malicious code detection scheme for trusted execution environments based on homomorphic encryption is established.

First, we implemented the experimental environment and verified the effectiveness of the detection algorithm. And then we analyze the system performance of the HE-TEEMD scheme.

7.1 Validity Verification

In the experiment, we compiled a trusted application in the secure world of OP-TEE, and a client application running in the normal world. We send a random integer a to the trusted application from the client application, such as 4973. Trusted application inverts the received integer data to an unsigned long integer b, where b=18446744073709546643. The trusted application encrypts a and b by Paillier homomorphic encryption and sends the ciphertext back to the client application. The client application multiplies the ciphertext, decrypts and verifies whether the result is a specific value to determine whether the two ciphertexts match.

In Fig. 6, the picture shows the output of the normal world and the secure world of OP-TEE, The results show that any integer and its negative numbers satisfy Corollary 1, the result of decrypting the product of its ciphertext is 0 on the integer domain. It takes 0.012 ms to perform a homomorphic computation and decryption operation in the normal world, where it costs 0.488 ms to perform an encryption operation in the secure world. When the database contains 3,801 signature, 1.27 seconds time-consuming will be generated to detect 100 bytes data, and 0.21 seconds for 50 bytes data. This means that HE-TEEMD enables transparent real-time detection to users.

The detection accuracy is an important indicator for evaluating malicious code detection mechanisms, the successful detection rate of HE-TEEMD is totally depended on the ability of the signature database. The detector download the ciphertext signature database from the trusted security

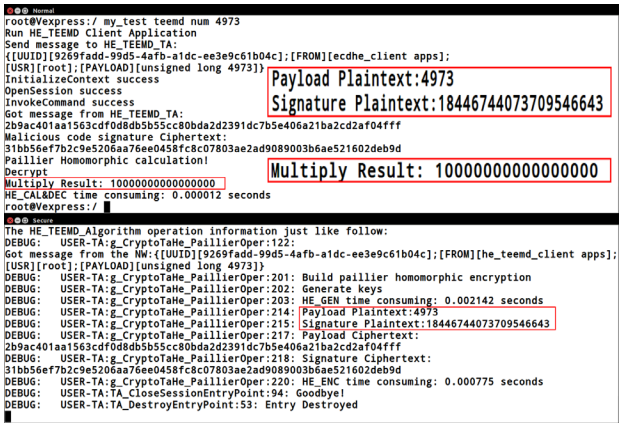


Fig. 6 The result of the validity verification.

Table 3 The result of performance comparison.

Scheme	TCB Inc. (KB)	Time Cons. (ms)	Storage Cons. (KB)	Update Cost
HE-TEEMD	1.4	244.7		NONE
LOCAL	248.9	0.3	26.7	HIGH

services provider, any signature that exists in the database will be detected 100%.

7.2 Performance Verification

After verifying the function of our method, we build a plaintext database containing 3801 rows of signature strings of arbitrary length in the normal world, and test the performance of the signature comparison operation under the HE-TEEMD scheme. In the experiment, we encrypt the plaintext information byte by byte, and find the character string fragment that can match the signature in the database by a brute-force search. In order to eliminate the influence of other factors, we do not perform other data operations in the experiment. We added time logging at the beginning and the end of the process and printed out the time delay. We implement a malicious code detection program that runs locally in the secure world (LOCAL). The LOCAL perform signature matching for plaintext directly. The data is compared mainly in terms of the time consumption and TCB growth. The comparison result is shown in Table 3.

As indicated in the table above, HE-TEEMD has apparent advantages in the increment of TCB generated. Unlike the LOCAL scheme, the TCB increment of HE-TEEMD scheme does not increase with the iteration of function and version. It should be pointed out that the HE-TEEMD scheme will generate a code increment of about 200 KB for homomorphic encryption. In our experiments, we simulate a switchable peripheral which means that the code increment for homomorphic encryption does not count toward the growth of the TCB. This is because the hardware chip does not meet the definition of Trusted Computing Base (TCB) for trusted OS which is neither a security protection mechanism for the secure OS nor increases the security risk of the secure world.

In terms of storage overhead, the LOCAL scheme needs to store all malicious code signature database in the secure world, which will result in corresponding storage overhead. On the other hand, we encrypt 50 bytes of plaintext information and perform the signature comparison operation with the 3801 line of arbitrary length signature strings, resulting in a delay of about 214 ms. Compared with the LOCAL scheme, the static malicious code analysis results in a certain time delay. Therefore, it is possible to real-time detect a tiny blob of data which is less than 1Kb, such as crossing-data between the two worlds. A free time mode is suitable for a bigger load such as programs and files analysis.

In order to test the impact of the introduction of evaluation variables on performance, we perform the signature comparison operation for different lengths of plaintext, and the test results are shown in Fig. 7.

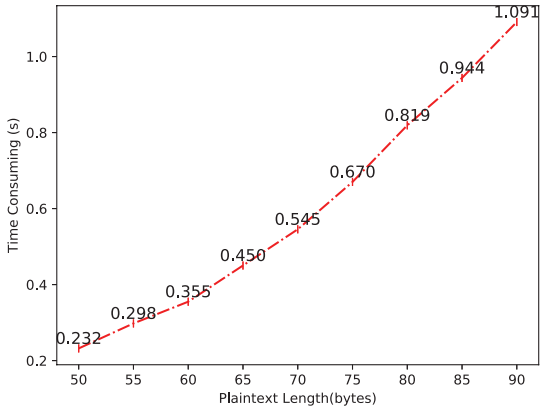


Fig. 7 The performance cost for different lengths of plaintext.

As shown in Fig. 7, when using a signature comparison algorithm with brute-force search, increasing the length of the plaintext to be detected will consume more time, which means that the added evaluation variable should be kept on a scale smaller than the length of the plaintext to be detected.

7.3 Security Analysis

We had validated the effectiveness of the malicious code detection scheme. In the end, we analyze the security of HE-TEEMD by discussing how it defeats security threats. As mentioned in Sect. 4, we assume that the components running in the TEE are entirely trusted, but the application installed by the user is considered suspicious. The threats of HE-TEEMD come from the attacks against the Detector in the Normal World and the attacks against the Communications between of the two worlds, so availability attack and masquerade attack are the primary security risk to HE-TEEMD.

Since the detection decisions are made by the detector in the normal world, the security of the detector is the cornerstone for the HE-TEEMD architecture to work properly. We achieved integrity availability and confidentiality by introducing the Evaluation Variables and the Homomorphic Encryption.

Integrity. A root privileged user is free to change the configuration and files of the user program, which can destroy integrity. As we mentioned in Sect. 6.3, we believe that any behavior that undermines the integrity of the detector will result in a deviation of the detection result. Assume that the data to be detected is $data_{content}$, which contains the malicious code fragment $block_{evil}$. We add the evaluation variables $data_{eval}$ to the detection queue, which also contains a malicious code fragment $block'_{evil}$. Since the randomness of the ciphertext of the homomorphic encryption, an attacker cannot distinguish between $data_{content}$ and $data_{eval}$, that is it is impossible to tamper or hide the $block_{evil}$ while outputting the correct $block'_{evil}$. Therefore, any behavior that destroys integrity will be discovered because of differences

in the detection results of the Evaluation Variables.

Availability. Availability attack is the biggest threat of the detector which is running in the normal world, an attacker can block and spoof the communication between the Detector and the secure world. However, Similar to the integrity attack, an attacker cannot identify the part of the detected content that belongs to the Evaluation Variables, nor can it create a Detector copy to output the correct detection result for the Evaluation Variables. On the one hand, when the attacker's intent is to obtain sensitive data from the secure world, blocking communication between the detector and the secure world will cause the attackers exposed themselves which is useless and not sensible. On the other hand, since the secure world has higher permissions, resetting the Detector program or even the normal world kernel can easily solve such attacks.

Confidentiality. The homomorphic ciphertext of the data to be detected has randomness, even if the attacker has the ciphertext of the data to be detected and some possible plaintext, the plaintext cannot be matched with the ciphertext. Therefore, the data to be detected is transparent to the normal world. On the other hand, the confidentiality of the detector program itself can be protected by a lot of perfect software protection mechanisms, such as confusion.

To sum up, any behavior that destroys the integrity and availability of the detector will manifest itself as an abnormal detection result to the Evaluation Variables. Since the secure world has higher privileges, it can operate on the detector program and the normal world kernel when an exception is detected, such as initialization the Detector program and the normal world system. Therefore, the security of the Detector and the communication is guaranteed.

8. Conclusion

This paper proposed an innovative application of the paillier homomorphic encryption technology to the TEE to achieve a malicious code detection scheme with a minimal TCB and interaction delay. The focus is to demonstrate the feasibility and security of the proposed architecture. The plaintext data is encrypted into ciphertext and outsourced to the normal world for malicious code detection with the randomness and homomorphism of the ciphertext of the homomorphic encryption algorithm. The feasibility of this scheme is verified by constructing the experiment environment. We further make performance evaluation, finding that the HE-TEEMD scheme has obvious advantages in terms of TCB increment, storage and update cost. At the same time, we found that with the increase of the payload to be detected, the time delay will also increase obviously. When performing complex detection tasks, the HE-TEEMD scheme can be used as a free-time task in addition to real-time detection. To the best of our knowledge, this is the first and only existing solution. In our future work, an advanced detection algorithm is the focus of our research work, which is more suitable for the HE-TEEMD and the signature matching algorithm.

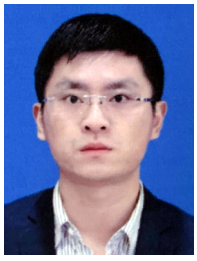
Acknowledgments

This work was supported by the National Natural Science Foundation of China (General Program) under Grant No. 61572253, the 13th Five-Year Plan Equipment Pre-Research Projects Fund under Grant No. 61402420101HK02001, and the Aviation Science Fund under Grant No. 2016ZC52030.

References

[1] G.D. Technology, "TEE Internal API Specification," http://www.globalplatform.org/specificationsdevice.asp, 2011. [Available Online;].
[2] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," Proc. 3rd ACM Workshop on Scalable Trusted Computing, STC'08, pp.21–30, New York, NY, USA, ACM, 2008.
[3] V. Costan and S. Devadas, "Intel SGX explained," Cryptology ePrint Archive, Report 2016/086, p.108, 2016.
[4] ARM, "ARM Security Technology: Building a Secure System using TrustZone Technology ARM," ARM white paper, p.108, 2009.
[5] T. Alves and D. Felton, "Trustzone: Integrated hardware and software security," ARM white paper, vol.3, no.4, pp.18–24, 2004.
[6] "Arm annual reports," http://ir.arm.com/phoenix.zhtml?c=197211&p=irolreportsannual, (Accessed on 10/01/2018).
[7] J. Yiu, "ARMv8-M Architecture Technical Overview," ARM white paper, vol.10, 2015.
[8] M. Lipp, D. Gruss, R. Spreitzer, C. Maurice, S. Mangard, M. Lipp, D. Gruss, R. Spreitzer, and S. Mangard, "ARMageddon: Cache attacks on mobile devices this paper is included in the proceedings of the ARMageddon: Cache attacks on mobile devices," USENIX Security, pp.549–564, 2016.
[9] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," 26th USENIX Security Symposium (USENIX Security 17), pp.1057–1074, 2017.
[10] E. Gallery and C. Mitchell, "Trusted mobile platforms," Foundations of Security Analysis and Design IV, pp.282–323, 2007.
[11] Z. Ning, F. Zhang, W. Shi, and W. Shi, "Position paper: Challenges towards securing hardware-assisted execution environments," Proc. Workshop on Hardware and Architectural Support for Security and Privacy (HASP), pp.1–8, 2017.
[12] F. Zhang and H. Zhang, "SoK: A study of using hardware-assisted isolated execution environments for security," Proc. Hardware and Architectural Support for Security and Privacy 2016 on - HASP 2016, pp.1–8, 2016.
[13] D. Rosenberg, "Qsee trustzone kernel integer over flow vulnerability," Black Hat conference, pp.1057–1074, 2017.
[14] A. Machiry, E. Gustafson, C. Spensky, C. Salls, N. Stephens, R. Wang, A. Bianchi, Y.R. Choe, C. Kruegel, and G. Vigna, "BOOMERANG: Exploiting the semantic gap in trusted execution environments," Proc. 2017 Network and Distributed System Security Symposium, 2017.
[15] "Bugs in htc's tee," http://atredispartners.blogspot.com/2014/08/here-be-dragons-vulnerabilities-in, (Accessed on 09/15/2018).
[16] "Cve-2016-3931-android open source project," https://source.android.com/security/bulletin/2016-10-01.html, (Accessed on 09/15/2018).
[17] "Cve-2015-4422 two privilege escalation vulnerabilities in huawei mate 7 smartphones," https://www.huawei.com/en/psirt/security-advisories/hw-432799, (Accessed on 09/15/2018).
[18] "Cve-2017-1000412," https://www.op-tee.org/security-advisories/, (Accessed on 09/15/2018).
[19] "Qsee privilege escalation vulnerability and exploit (cve-2015-6639)," http://bits-please.blogspot.com/2016/05/qsee-privilege-escalation-vulnerability.html, (Accessed on 09/15/2018).

[20] "Github - op-tee/optee_os: Trusted side of the tee," https://github.com/OP-TEE/optee_os, (Accessed on 09/19/2018).
[21] "Globalplatform tee internal core api specification v1.1.2," https://globalplatform.org/specs-library/tee-internal-core-api-specification-v1-1-2/, (Accessed on 09/15/2018).
[22] S.D. Yalew, G.Q. Maguire, S. Haridi, and M. Correia, "T2Droid: A TrustZone-based dynamic analyser for Android applications," 2017 IEEE Trustcom/BigDataSE/ICESS, 2017.
[23] X. Zheng, Y. He, J. Ma, G. Shi, and D. Meng, "TZ-KPM: Kernel protection mechanism on embedded devices on hardware-assisted isolated environment," High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS), 2016 IEEE 18th International Conference on, pp.663–670, IEEE, 2016.
[24] M. Albrecht, M. Chase, H. Chen, J. Ding, S. Goldwasser, S. Gorbunov, S. Halevi, J. Hoffstein, K. Lauter, S. Lokam, et al., "Homomorphic encryption standard," 2018.
[25] D. Okunbor and C. Sarami, "Homomorphic encryption: A survey," Review of Business and Technology Research, vol.14, no.1, pp.1941–9414, 2017.
[26] G. Tuvell and D. Venugopal, "Malware detection system and method for mobile platforms," US Patent 9,104,871, Aug. 11 2015.
[27] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pp.223–238, 1999.
[28] R.A. Al-Shibib, Performance Analysis for Fully and Partially Homomorphic Encryption Techniques, Ph.D. thesis, Middle East University, 2016.



Ziwang Wang received the M.S. degrees in computer science and technology from Guizhou Normal University, Guiyang, China, in 2016. He is currently a Ph.D. candidate of the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics in China. His research includes mobile security and system security.

Yi Zhuang graduated from the Department of Computer Science, Nanjing University of Aeronautics and Astronautics in 1981. Now she is a professor and Ph.D. supervisor of the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics. Her research interests include network distributed computing, information security and dependable computing.