Hindawi
Security and Communication Networks
Volume 2020, Article ID 1756130, 16 pages
https://doi.org/10.1155/2020/1756130

Research Article

TZ-MRAS: A Remote Attestation Scheme for the Mobile Terminal Based on ARM TrustZone

Ziwang Wang, Yi Zhuang, and Zujia Yan

Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210008, China

Correspondence should be addressed to Yi Zhuang; zy16@nuaa.edu.cn

Received 27 October 2019; Revised 7 July 2020; Accepted 7 September 2020; Published 23 September 2020

Academic Editor: Leonardo Mostarda

With the widespread use of mobile embedded devices in the Internet of Things, mobile office, and edge computing, security issues are becoming more and more serious. Remote attestation, one of the mobile security solutions, is a process of verifying the identity and integrity status of the remote computing device, through which the challenger determines whether the platform is trusted by discovering an unknown fingerprint. The remote attestation on the mobile terminal faces many security challenges presently because there is a lack of trusted roots, devices are heterogeneous, and hardware resources are strictly limited. To ARM's mobile platform, we propose a mobile remote attestation scheme based on ARM TrustZone (TZ-MRAS), which uses the highest security authority of TrustZone to implement trusted attestation service. Compared with the existing mobile remote attestation scheme, it has the advantages of wide application, easy deployment, and low cost. To defend against the time-of-check-to-time-of-use (TOC-TOU) attack, we propose a probe-based dynamic integrity measurement model, ProbeIMA, which can dynamically detect unknown fingerprints that generate during kernel and process execution. Finally, according to the characteristics of the improved dynamic measurement model, that is, the ProbeIMA will expand the scale of the measurement dataset, an optimized stored measurement log construction algorithm based on the locality principle (LPSML) is proposed, which has the advantages of shortening the length of the authentication path and improving the verification efficiency of the platform configuration. As a proof of concept, we implemented a prototype for each service and made experimental evaluations. The experimental results show the proposed scheme has higher security and efficiency than some existing schemes.

1. Introduction

With the popularity of the Internet of Things, edge computing, and FIDO (Fast IDentity Online) devices, more and more enterprises are launching different devices to provide services to customers. The security-critical and privacy-sensitive data stored on these devices is also increasing, and the security of mobile devices has caused great concern in the industry and academia. Currently, ARM processors occupy the processor market for most mobile and embedded devices, providing more than 60% support for all embedded devices and more than 4.5 billion mobile phones [1]. Among them, ARM processor chips account for more than 95% of the processor chips of smart mobile phones, and ARM equipment is estimated to reach 1 trillion by 2035 [2]. However, the security risks of ARM terminals have become an essential factor in limiting their development, especially the application and popularization of smartphones and the Internet of Vehicles. Untrusted terminals may bring severe security and privacy problems to enterprises and individuals. ARM TrustZone is a hardware solution for trusted computing [3], which can produce a highly secure isolation environment that provides users with data, firmware, and peripheral security. Profiting from the fact that TrustZone technology has been applied to most mobile embedded devices, TrustZone has become the de facto standard for mobile devices to implement trusted computing, providing trusted security services. According to statistics, about 1 billion wearable devices, mobile devices, industrial devices, and other IoT devices used TrustZone-based trusted execution environments in 2017 [4].

At present, technologies to improve the security of mobile devices include intrusion detection, malicious code analysis, and virtualization. However, these techniques are based on the same assumption that the mobile operating system kernel or the hypervisor is trusted and that the technology itself does not guarantee the trustworthiness of the entity. The remote attestation can effectively solve this problem to a certain extent. According to the remote attestation framework defined by the Trusted Computing Group (TCG), the current implementation of the standard remote attestation (RA) is based on integrity measurement, and the measurement result for the remote attestation is stored in an isolated trusted component called the trusted platform module (TPM) [5]. To extend the RA to mobile embedded devices, TCG proposed a specification for mobile trusted modules (MTM) [6]. The specification supports the implementation of the TPM as an integrated solution or firmware using processor technologies such as the trusted execution environment (TEE). The TEE quickly becomes the primary trusted root for mobile embedded devices, surpassing other security solutions on resource-constrained devices [7], e.g., secure element (SE). This means that a software trusted platform module (STPM) can be implemented in the TEE's secure environment, with security provided by the TEE's hardware mechanisms, providing practical, secure, and interoperable standard TCG access for mobile devices.

Remote attestation is a process of verifying the integrity status and credibility of a remote computing device and a universal protocol accepted and used by most entities [8]. The implementation of the remote attestation is usually divided into static attestation and dynamic attestation. Most existing attestation techniques are based on binary fingerprints of system memory called the binary-based remote attestation. Most binary-based approaches are the static attestation that collect device integrity information at a particular time. A dynamic attestation refers to the process of dynamically collecting device integrity information without interrupting normal operations, including the collection and analysis of multidimensional information such as control flow, system calls, and memory fingerprints. That is, a well-designed binary-based approach can also implement dynamic attestation. Generally, dynamic attestation requires higher energy consumption and computational resources.

In the binary-based approach, a terminal device needs to provide all platform configuration information to obtain the trust of the remote platform, including the detail information of the operating system and application software. The security vulnerability information of the device will be leaked out and become the target of hacker attacks. Second, a large amount of user software and system software makes the scalability worse, which is the main bottleneck of the binary-based approach. The existing research gets better protection of privacy by using Merkle hash trees [9] instead of the linear storage solutions. However, implementing the Merkle hash tree-based approach in embedded devices encounters more severe challenges due to the strictly limited performance.

1.1. Challenges. Therefore, implementing a well-performance remote attestation for the resource-constrained mobile embedded devices is not a trivial matter, which mainly has the following three challenges: (1) TPM is often not available in embedded systems [10]. The significant challenge is how to build the root of trust for measurement (RTM), the root of trust for storage (RTS), and the root of trust for reporting (RTR). (2) The binary-based remote attestation approach can implement to the resource-constrained devices with small overhead [11, 12]; how to design an effective defense against TOC-TOU attacks for the binary-based remote attestation is another critical challenge. (3) The stored measurement log (SML) based on the hash tree is a solution generally accepted by researchers. However, the measurement data saved in the form of a hash tree will generate a more considerable overhead in the construction and update phase. So, how to optimize the performance of SML construction and updates is also a challenge.

With the trend of integrating security cores and trusted application (TA) in commercial mobile devices, TrustZone will be the best choice for the trusted remote attestation of heterogeneous multiplatform mobile devices based on the ARM processor. We found it practical to use the STPM run in the secure world to provide a trusted root service. Raj et al. [10] implemented a fully functional trusted root module with TrustZone, named fTPM, and gave solutions such as trusted roots, trusted storage, security counters, and security clocks. Based on [10], this paper develops the remote attestation system components with ARM TrustZone and implements it on the hardware platform. We simulated the core function modules such as secret key algorithm compute unit, tamper-resistant certificate storage unit, integrity measurement root and report root, and built a remote attestation system for ARM embedded devices. We implement the required functionality, which is usually supported by the TPM, given a solution to the challenges of missing TPM chips in embedded devices.

The second challenge is the same for all binary-based attestation schemes. We designed and proposed a dynamic SML update mechanism based on the probe mechanism rather than a complex TOC-TOU consistency check mechanism. The binary-based approach captures the measurement values of the entity during program loading, reflecting the state of the code and data at the time the measurement occurred, but a runtime state. If the system is vulnerable, the attacker can use the TOC-TOU defects to create an attack, that is, affecting the data fingerprint collected during the measurement execution by attacking the measurement strategy, manipulating the measurement result calculated by the prover. We proposed a probe-based system kernel and program integrity dynamic monitoring model (ProbeIMA). Any writes or changes to memory fragments that affect the program and kernel's integrity will generate a new integrity measurement record and the changes to essential registers that affect the application and kernel's integrity.

The Merkle hash tree is a fundamental technology in remote verification, which is flexible and protects privacy. Many research studies have proposed to assign different weights to the hash tree leaf nodes to achieve a better model [13, 14]. However, the prior has not offered a feasible algorithm to compute and update the weight of each leaf node. This paper proposes a locality principle-based hash tree construction strategy and designs a complete construction algorithm, providing a low-cost integrity SML solution for mobile devices.

1.2. Contributions. Aiming to implement remote credibility verification on the resource-constrained mobile terminal, we propose a TrustZone-based remote attestation scheme (TZ-MRAS), perform real-time integrity monitoring, and maintain the SML updates with low cost. In summary, our contributions in this paper are as follows:

(i) We improve the existing trusted remote attestation framework based on the trusted execution environment. A software TPM is implemented on mobile devices to provide a trusted root for the remote attestation's measurement, report, and store the module. It offers numerous advantages over the mobile device in terms of lower cost, wide application range, and ease of deployment.

(ii) We propose ProbeIMA, an approach that uses the probe-based mechanism to implement dynamic SML update, which provides an effective solution for achieving the binary-based remote attestation scheme against TOC-TOU attacks.

(iii) We propose an unbalanced hash tree update and verification optimization scheme and give the algorithm of construction and update. The experimental results show it has the advantages of shortening the length of the attestation path and improving efficiency.

(iv) We build the prototype of the proposed remote attestation scheme, realize, and verify the effectiveness and superiority of the proposed algorithm and model.

The remainder of the paper is organized as follows: we offer related work in Section 2. Section 3 introduces relevant background knowledge. Section 4 presents the proposed remote attestation model and describes the design of ProbeIMA and LPSML. Section 5 details the design and implementation of our approach. Section 6 describes the implementation and evaluation of our prototype system and presents the results and analysis. Finally, in Section 7, we conclude our work.

2. Related Work

The remote attestation of the terminal device is essential for mobile security and IoT security [3, 11, 12, 15]. In the mobile computing scenario, the terminal needs to prove to the remote server software is running on the device and whether their integrity is destroyed. The core issue of the mobile remote attestation is to study how to implement the trusted root construction and to design a lightweight remote attestation mechanism for the embedded platform. On the one hand, due to the different hardware characteristics of the mobile platform, the existing research mainly focuses on the design of the trusted root implementation of the ARM platform and studies ARM's integrity protection mechanism (including trusted measurement and trusted storage mechanism). On the other hand, the performance cost of the SML usually exceeds the carrying capacity of the embedded platform. The performance optimization of the SML is also the focus of the current research.

2.1. The Implementation of the Trusted Root and IMA for ARM. TPM-based integrity attestation uses the TPM as the root of trust and leverages the microprocessor within the TPM to provide key calculation and integrity measurement services. The TPM can sign and store the measurements produced by the integrity measurement component. However, due to limitations in hardware size and energy consumption of the embedded system, TPM hardware for the mobile platform has not been put into practical production and deployment. Therefore, the TPM mobile reference architecture [16] abstracts the TPM chip functionality into a higher-level concept called a protected environment or a trusted execution environment that has been deployed into all of the popular hardware platforms. [16, 17] discussed the scheme of simulating the TPM based on the embedded processor's TEE and gave a positive result.

The integrity measurement architecture (IMA) [18] was first proposed by IBM. It refers to that, during the application execution and the dynamic link library or kernel loading, the components in the kernel perform a measure on the critical data, such as the program code and the configuration files, construct, and store the list of the measurement output. The IMA is able to extend the Trusted Boot of TCG to the system's application layer and build a complete chain of trust from the TPM to application software. Samsung first proposed to deploy IMA to a trusted execution environment and realized the design of the industrial-grade architecture, namely, TIMA [19], which used the ARM TrustZone hardware architecture to achieve secure and reliable measurement and report module. TIMA performs continuous integrity monitoring to the Linux kernel, that is, the components of TIMA regular scans of the kernel code segments and other critical data to ensure that the REE is not tampered with.

In addition, the strategies for implementing the terminal device integrity attestation include hypervisor-based technologies [20] and coprocessor-based technologies [21]. These mechanisms cannot offer the highest-level remote integrity attestation solutions due to the lack of support for hardware trusted roots.

2.2. Enhanced Mobile Platform Integrity Protection. HIMA is a hypervisor-based integrity measurement architecture that uses a separate administrator GVM (guest virtual machine) to store the measurement data for other GVMs [22]. The VMM provides isolation between measurement targets and measurement agents and proactively monitors critical events in the goal GVM to avoid TOC-

TOU attacks. Similarly, SIMA [23] uses a different approach which places the sensor agent into the GVMs as a kernel module. The sensor agent communicates using a shared memory-mapped segment called the Blackboard. However, as described above, these approaches to using virtual hypervisors for kernel integrity protection represent a security risk. Therefore, many studies have attempted to implement a more secure and reliable integrity protection approach in an isolated execution environment based on TrustZone.

Initially, researchers used TrustZone's highest privilege to host the kernel integrity monitor and designed the basic architecture to protect the OS kernel with a security monitor which is completely within the ARM TrustZone security realm. However, it does not provide real-time monitoring to the target kernel [24]. Furthermore, some critical functions are deprived from the kernel of the REE in TZ-RKP, such as the system control instructions and the update capability of memory translation tables, routing these functions to the secure world to effectively block the attacks of unauthorized modifications or binary injections and ensure the integrity of the kernel.

Similarly, Sprobes [25] placed probes to the filtered system instructions and dynamically interrupted the target transactions in the normal world by using the higher-privilege secure world, which implements a better integrity protection scheme.

2.3. SML Performance Optimization. The Merkle hash tree was first introduced by Merkle to solve the problem of Public-Key Infrastructure (PKI) certification [9]. The Merkle hash tree is a powerful solution to store large amounts of data in minimal storage spaces. In the trusted computing field, the traditional integrity verification framework needs to measure and verify all running program modules, which may present issues such as privacy leaks and low efficiencies. Xu et al. [26] proposed a remote attestation mechanism based on the Merkle hash tree in which some module measurements are hidden and do not need to be reported. To some extent, this method can avoid the privacy leak of the prover and improve the efficiency of verification but has a low construction efficiency.

To improve the construction efficiency of the hash tree, Zhu et al. [27] presented an integrity measurement model based on the left-full tree (LFT). Although the hash tree's construction is optimized and its time consumption is reduced, the model failed to achieve an appropriate verification efficiency of the remote attestation. Fu et al. [13] proposed a construction method UBTS-SML based on the unbalanced tree, which improved the verification efficiency of the remote attestation. However, this method needs to update the structure of the tree for each application request. The update and maintenance of the hash tree are inefficient, and the revocation of leaf nodes is not taken into consideration, which may result in the depth of the hash tree being too large.

In summary, aiming at the shortcoming of the mobile terminal integrity dynamical measurement and the lightweight SML mechanisms, we present a TrustZone-based innovation framework called TZ-MRAS, which has higher security and performance.

3. Preliminaries

3.1. ARM TrustZone. ARM TrustZone is the hardware solution for trusted computing on ARM devices. TrustZone framework uses the trusted bootloader stored in independent read-only memory (ROM) as the trusted root, implements authentication and initialization of trusted components to create a complete chain of trust, and guarantees the security of the entire system. ARM TrustZone shares the processor into two separate operating environments in a time-sliced manner, the normal world and the secure world. They are also known as the rich execution environment (REE) and the trusted execution environment (TEE). A new bit called nonsecure bit (NS) of the Security Configuration Register (SCR) is used to determine in which world the current processor is running, which can also extend the isolation to other hardware resources such as memory, cache, and controllers.

The architecture of ARM TrustZone is shown in Figure 1. The TrustZone technology defines two different executed contexts independently, and the ARM CPU has separate register banks for each of the two worlds. In general, the secure world has higher privileges than the normal world, and the TrustZone architecture guarantees that the trust chain can be delivered to the kernel and the trusted application (TA) of the secure world. TrustZone also includes the secure monitor, which is a specialized processor mode for the ARM CPU, that saves the context of the current world and restores the context at the location it switches. The ARMv8 architecture introduces the exception level (ELx) to indicate the processor's permissions. The greater the level indicator x, the higher the executive privilege. The highest privileged mode EL3 is given to the secure monitor. The ARM instruction called SMC (secure monitor call) and the secure interrupts are used as the sources to trigger the EL3 exception, that is, the normal world triggering to the monitor mode by EL3 exceptions. The program in the monitor mode saves the CPU context of the nonsecure mode and then sets the NS bit to 0, indicating that the secure world is triggered and can offer the security services provided by the trusted component of the secure world. In return, the NS bit is set to 1, and the CPU context of the nonsecure mode is restored.

This paper implements a scheme to build a trusted root for the mobile remote attestation. The chain-of-trust can be trusted delivery to the security modules of the TEE through the ARM's hardware protection mechanism, which guarantees the software TPM obtain the same trustworthiness to dedicated trusted hardware.

3.2. Locality Principle. Tree-formed verification data can effectively solve the privacy protection problem of integrity attestation. However, constructing and updating tree-formed data require more computing resources. This paper

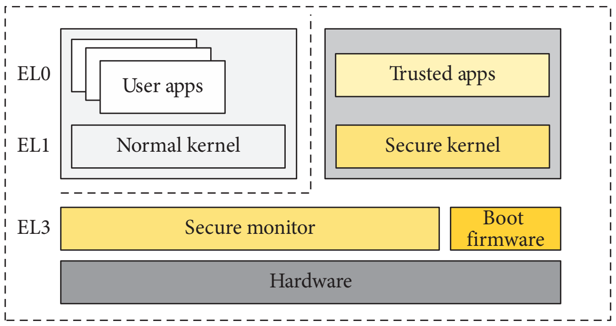


Figure 1: The architecture of ARM TrustZone [28].

introduces the locality principle to optimize the performance of the tree-formed verification data.

Locality principle refers to the fact that when researchers analyze the running program, they found that data access is phased and shows a phenomenon of aggregation [29]. This aggregation is usually reflected in two dimensions: the locality of time and the locality of space. The locality of time means that data is likely to be revisited soon after it is accessed. The locality of space means that the adjacent data may also be accessed soon.

In the measurement phase of remote attestation, the program's integrity measurements need to be stored in the SML. Due to the dynamic nature of the program, it is usually necessary to update the SML. The locality of time is reflected here that when a program's integrity measurements are updated to the SML for requesting service, it is likely to request the service again soon. Making use of this feature to construct the SML can enhance the verification efficiency of the remote attestation.

Therefore, we define the strength of the locality principle in the RA as follows: suppose at the time interval $t_1, t_2, \ldots, t_n$, the set of programs for requesting services is $s_1, s_2, \ldots, s_n$. IOS_n represents the intersection of the request set between t_{i-1} and t_i, while UOS_n represents the union set, which is

$$\mathrm{IOS}_n = s_{n-1} \cap s_n, \mathrm{UOS}_n = s_{n-1} \cup s_n. \quad (1)$$

Let $\mathrm{card}(\cdot)$ denote the number of elements in the set; the intensity of locality (IOL) can be defined as equation (2) and satisfy Theorem 1 and Theorem 2:

$$\mathrm{IOL}_i = \frac{\mathrm{card}(\mathrm{IOS}_i)}{\mathrm{card}(\mathrm{UOS}_i)}. \quad (2)$$

Theorem 1. *The range of IOL_i is [0, 1].*

Proof of Theorem 1

$$\mathrm{IOL}_i = \frac{\mathrm{card}(\mathrm{IOS}_i)}{\mathrm{card}(\mathrm{UOS}_i)} = \frac{\mathrm{card}(s_{i-1} \cap s_i)}{\mathrm{card}(s_{i-1} \cup s_i)},$$
$$0 \le \frac{\mathrm{card}(s_{i-1} \cap s_i)}{\mathrm{card}(s_{i-1} \cup s_i)} \le \frac{\mathrm{card}(s_i)}{\mathrm{card}(s_i)} = 1. \quad (3)$$

□

Theorem 2. *The value IOL_i is positively related to the number of programs requesting the service again at t_i.*

Proof of Theorem 2

$$\begin{aligned}\mathrm{IOL}_i &= \frac{\mathrm{card}(\mathrm{IOS}_i)}{\mathrm{card}(\mathrm{UOS}_i)} = \frac{\mathrm{card}(s_{i-1} \cap s_i)}{\mathrm{card}(s_{i-1} \cup s_i)} \\ &= \frac{\mathrm{card}(s_{i-1} \cap s_i)}{\mathrm{card}(s_{i-1}) + \mathrm{card}(s_i) - \mathrm{card}(s_{i-1} \cap s_i)} \\ &= \frac{1}{(((\mathrm{card}(s_{i-1}) + \mathrm{card}(s_i))/\mathrm{card}(s_{i-1} \cap s_i)) - 1)}.\end{aligned} \quad (4)$$

And when $\mathrm{card}(s_{i-1} \cap s_i) > \mathrm{card}(s_{i-1} \cap s_i)'$, $\mathrm{IOL\_}i > \mathrm{IOL}_i'$.

The above theorems show that when the number of programs requesting the service again is larger, the value of IOL_i is closer to 1, and accordingly, the verification performance of the remote attestation mechanism based on the locality principle is better. □

4. Design

4.1. Threat Model and Assumptions. We follow the assumptions in the TrustZone model that the processor is trusted, and the ARM hardware security mechanism can extend the chain of trust to trusted applications, that is, TAs running in the secure world are trusted, and the client applications (CAs) running in the normal world are untrusted but can be measured.

The secure world is the trusted computing base (TCB) of our approach, and we can use the programs in the secure world to measure the integrity of the normal world. We assume that a subset of TPM functional modules that meet the trustworthiness requirements has been simulated in the TEE, such as the secure storage, real-time clock, anticollision hash functions, and cryptographic modules. We assume that there is secure communication between the prover and the verifier (the initiator and responder of the attestation request). We take the kernel-level attackers who can destroy system kernels and components running in the REE into consideration. According to the TPM 2.0 specification, the denial-of-service attack (DoS attack) is out of scope. As long as the untrusted kernel can access the mobile device resources, there will be a DoS attack. However, the TEE can check the DoS by actively querying the access record of the remote server and then initialize or sanitize the compromised REE. So, we do not consider the DoS attack. Finally, we do not consider the side-channel attack.

4.2. The Design of the TZ-MRAS. As described in Section 1.1, there are three main challenges to the mobile remote attestation system. Aiming at these challenges, we have studied and proposed a TrustZone-based mobile remote attestation scheme (TZ-MRAS). First, we introduce the model overview of the remote attestation architecture based on ARM TrustZone. To evaluate the mobile device's state of trustworthiness, we measure the integrity of the kernel and CAs in the REE by the RTM and the RTR deployed in the secure world. Second, we design a binary-based integrity measurement mechanism ProbeIMA which can withstand the TOC-TOU attack. Third, we use LPSML to optimize the construction and update the algorithm for SML. In summary, this paper presents three valid mechanisms to implement the static integrity attestation scheme to address the challenge that are a remote attestation model based on the TEE, a probe-based integrity measurement mechanism (ProbeIMA), and an LPSML-based measurement data optimization mechanism.

4.2.1. The Remote Attestation Model Based on the TEE. A standard TCG remote attestation process can report the device's environmental fingerprint to the verifier and attest the identity and trustworthiness status of the prover, which includes an identity attestation process and a platform integrity attestation process. Because of space constraints and the current direct anonymous attestation-based identity authentication is mature and more universal with the PC devices, we omit the process of identity authentication.

Remote attestation scheme based on the TEE is not new, most of which is based on the Intel's SGX and the TrustZone. However, in the mobile field, both can only offer static remote attestation services but cannot address the TOC-TOU attack. Generally, when the measurement module in the RA receives an attestation request, it will switch to the TEE, and the normal execution environment will be paused. Then, the measurement module computes the normal world fingerprint from the storage unit at the trigger time and sends the results to the attestation challenger.

Figure 2 provides an overview of our architecture, using ARM TrustZone security extension to implement remote attestation services that can cope with TOC-TOU attacks. As shown in Figure 2, the required remote attestation modules based on the TEE are foundational of TZ-MRAS that make a credibility verification process valid and trusted. Besides, two additional approaches are proposed for building a remote attestation framework for mobile terminals: the ProbeIMA and the LPSML.

First, our scheme solves the first challenge that lacks the TPM chip in embedded devices by implementing the standard TCG TPM functionality using the ARM TrustZone-based TEE. We build a software TPM as the root of trust for remote attestation components, providing all TPM functions, including secure storage and key operations. Specifically, we use trusted applications running in the TEE to simulate the trusted measurement module and report module that are isolated from the normal world. The measurement module acts as a watch program running in the secure world, which can not only dynamically detect the integrity changes of the kernel and the program but also can update the SML. The report module acquires the current and historical fingerprints of the device and sends them to the attestation challenger. Finally, the challenger makes a remote attestation decision through the discovery of the unknown fingerprint.

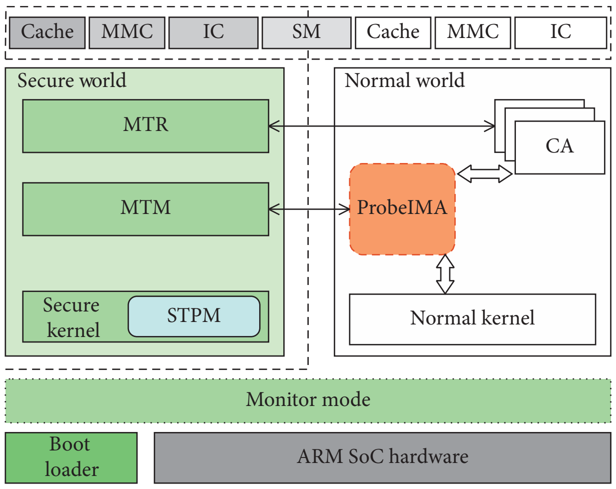


Figure 2: TEE-based mobile remote attestation architecture.

Second, we present a probe-based integrity measurement mechanism called ProbeIMA, which dynamically monitors the integrity change of the measurement object through the placement of the probe.

Therefore, TZ-MRAS, which does not rely on the dedicated TPM chip, can be deployed on any ARM-based mobile platform. It should be pointed out that, due to the introduction of the ProbeIMA, the SML includes the measurement records at the measured moment, as well as some historical measurement records caused by the integrity changes of the program and kernel generated by two measurement gaps. That is, a higher performance scheme of SML construction and update is necessary. To solve this problem, we design an SML structure model based on the locality principle, which can significantly increase the efficiency of SML construction and update.

4.2.2. The Design of the Probe-Based Integrity Measurement Architecture. In the binary-based remote attestation process, the challenger obtains the platform's integrity measurement record through the integrity challenge agreement and determines whether the platform is trusted by detecting the unknown fingerprint in the SML. Generally, the static attestation only obtains the information of the unknown fingerprint at the current moment; that is, the measurement module only verifies the integrity state at a particular moment. However, an attacker who gained the root access can dynamically tamper with the data according to the measurement granularity and manipulate the binary fingerprint acquired by the MTM, which means there is a race condition defect.

This paper proposes a probe-based SML dynamic update mechanism, ProbeIMA. By rewriting the binaries of the kernel and program code, probes are added at all of the instructions which can change the value of the key registers and the content of memory. Therefore, ProbeIMA can dynamically detect the generation of unknown fingerprints during system operation, extending the scope of the measurement module to the entire life cycle of the system's operation.

Because ARM has fixed-length instructions, it is easy to find the target instructions in the binaries. The flowchart of the probe-based integrity measurement architecture is shown in Figure 3. When a probe instruction placed in the binaries of the kernel or program is triggered, the secure world extracts the unforgeable state of the normal world from its hardware register directly, running the measurement module to analyze the instruction, for example, determining which program the instruction belongs to. Then, the ProbeIMA updates the integrity measurement record corresponding to the program ID. We count all the sets of page tables that affect the integrity of the kernel and the CAs, writing them to a specified secure memory address, called integrity-related memory address set (IMASet).

In addition, adding probes to the target instructions throughout the kernel and the entire CAs will cause the secure world to be frequently woken up. To solve this problem, we rewrite the code in the monitor mode to implement two processing logics as follows: (1) for the switch operation triggered by the probe before the kernel/program startup is completed, return to the normal world directly and execute the next instruction. This is because the initialization of the key registers during the kernel/program startup process will generate a large number of false triggers. Therefore, ProbeIMA will not cause user-aware latency during startup. (2) For all probes in the memory write instructions, obtain the physical address of the trigger instruction and determine whether it is included in the IMASet. If contained, obtain the corresponding program ID in the IMASet, and update its integrity record. The IMASet will significantly reduce the number of trigger instructions which is truly related to integrity and reduce performance consumption. As shown by the solid path in Figure 3, we refer to the trigger in the ProbeIMA that switches to the secure world and performs an integrity update as a valid trigger. Triggers that fall back to the normal world directly from the monitor mode are called invalid triggers. The operation logic and implementation method of the probe placement and program process are described in detail in Section 5.1.

4.2.3. The Locality Principle-Based SML Optimization Mechanism.

As mentioned above, the program's configuration modifications, the version alteration, and the execution of unknown programs will produce unknown fingerprints. Unlike the standard binary-based integrity attestation solutions, the latest kernel and program fingerprints do not immediately overwrite the previous records. This means that, in the ProbeIMA, sometimes, a program can have more than one fingerprint stored in the SML, which increases the performance consumption of the SML in the Merkle tree. This paper proposes an LPSML-based measurement data storage mechanism. First, the LPSML uses a hardware-protected register to store the root node of the SML tree, and the complete measurement data is stored in the secure memory of the secure world. So, any nonphysical attack (including malware, trojans, and others) cannot threaten the security and availability of the SML. Second, we design the corresponding construction, insertion, and revoke algorithms to avoid too frequent construction and update operations, increase the scalability of the SML, and improve the system performance. The specific construction and update algorithms are described in detail in Section 5.2.1.

4.3. The TZ-MRAS Remote Attestation Protocol.

In the TZ-MRAS, the sensitive module of the remote attestation includes the measurement module, the secure storage module, and the report module, which are isolated in the TEE to ensure that the ARM terminal trust chain can be passed from the root of trust to the sensitive module. Therefore, the TZ-MRAS protocol's core is to ensure that the verifier can detect it when the crucial memory or configuration of the prover is modified.

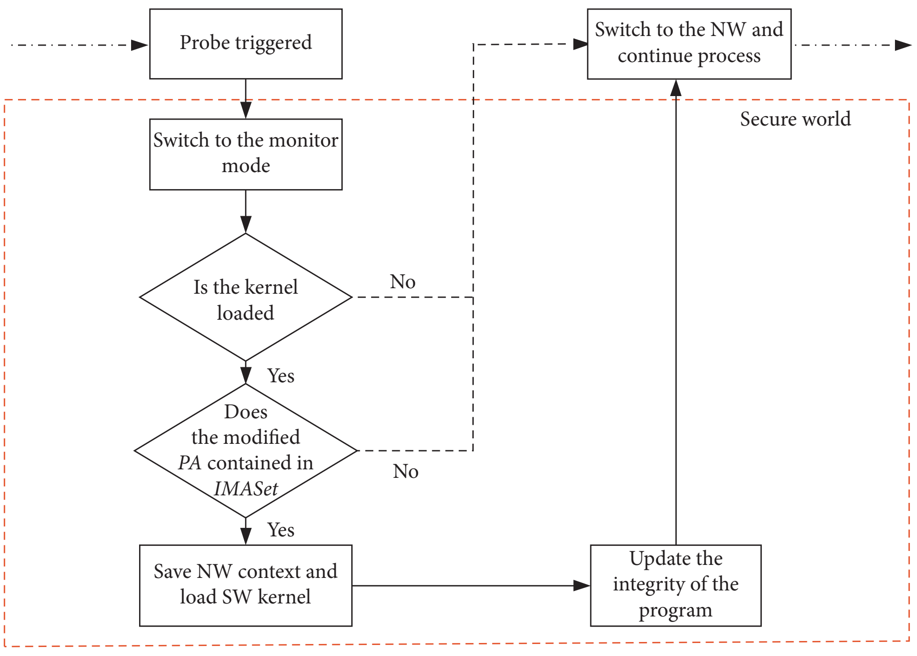
Figure 4 shows the TZ-MRAS protocol and the process of interaction between its components. In the protocol process, the integrity measurement and storage operation strictly limit executing in the trusted areas. The sensitive data is passed through the ciphertext; thus, the trust relationship between the prover and the verifier is built on the cryptographic algorithms. The TZ-MRAS remote attestation protocol involves the following steps:

(i) Step 1: the attestation agent of the verifier generates a random number *nonce* and prepares to send it to the prover's attestation agent which is named RA_Router.

(ii) Step 2: RA_Router receives and sends the challenge message to the trusted kernel.

(iii) Step 3: the report module MTR loads the identity verification key AIK, serializes the request service program ID with the random number *nonce*, and signs and reports the measurement information by using the software trusted platform module, which includes the following: (1) the measurement module reads the root hash value of the platform configuration which is contained in the (platform configuration register) PCR register, serializes it with the random number *nonce*, and signs it within the software trusted platform module. (2) The measurement module generates the attestation path of the request-target and sends it and the results in (1) to the report module. (3) The report module signs and sends the required data including each nodes' hash of the attestation path, the PCR value, and the signed challenge message to the verifier.

(iv) Step 4: The Verifier verifies and replies to the attestation data according to the reference data. The details are as follows: (1) the attestation agent sends the attestation data to the verification agent. (2) The verification agent checks the random number *nonce* and the AIK signature to verify the legality of the prover's identity. Besides, *nonce* can also be used to verify the freshness of the attestation conversation. (3) The verification agent queries the standard value of the local reference data, compares each node's


Probe triggered
Switch to the NW and continue process
Secure world
Switch to the monitor mode
Is the kernel loaded
No
Yes
Does the modified *PA* contained in *IMASet*
No
Yes
Save NW context and load SW kernel
Update the integrity of the program

PA	physical address;	- - ▶	invalid trigger;
NW	normal world;	⟶	valid trigger;
SW	secure world;	·-·▶	instruction processing;
IMASet	set of memory page that affects integrity;		

FIGURE 3: Flowchart of the ProbeIMA.

hash with the references, and compares the root hash which was recalculated by using the attestation path with the PCR value. If it is consistent, the integrity attestation is passed.

5. Implementation

5.1. The Probe-Based Dynamic Measurement Mechanism. The ProbeIMA is a runtime integrity measurement mechanism that extends the detection of unknown fingerprints to the entire life cycle of the system. The system environment fingerprint is updated by triggering the probe placed in the binary execution program. As the dynamic measurement component's entrance, the SMC instruction can distinguish different types via the command ID set in the regular register. On the contrary, the SMC instructions inserted as probes are fixed and will not cause the program integrity changes randomly, which is important for integrity management.

The ProbeIMA provides real-time monitoring for kernels and programs running in the normal world through the placement of probes. It mainly monitors the following two aspects: (1) detecting the execution of new programs; (2) the integrity change of the executing programs and kernels. First, detect the changes in the critical CPU registers that affect kernel integrity, such as SCTLR, TTBR, and TTBCR. Second, monitoring binary modification stored in the memory, that is, insert the SMC instructions at all instructions that can affect the integrity of the process's binaries, for example, the memory write instructions STR, STM, and SWP. When one of those inserted instructions is triggered, the handler running in the secure world performs an integrity measurement on the program which the physical memory address belongs to. It means that all instructions that affect the page tables' content will be routed to the secure world for handling. The integrity of the kernel or program will be updated based on the handle results of the trigger instruction.

5.1.1. Probe Placement for Key Registers. In the widely used ARMv7 architecture (32 bit), the control instructions are achieved by writing to the coprocessor registers. Because ARM enforces instruction alignment and fixed-length instruction [24], it is easy to generate the template for all MCR and LDC instructions which write to the coprocessor register. As shown in Figure 5, we place a probe after the MCR instruction which writes a value to the CP15 coprocessor register, causing the normal world to switch to the secure world to perform the integrity measurement. Querying whether the target register affects the integrity of the user environment, a filtered register bit set is also stored in the IMASet. The MTM recalculates the fingerprint of the instructions, whose memory address is stored in the IMASet, and generates a new integrity record. It should be pointed out that writing of some registers may affect the contents of the IMASet; that is, the content of the IMASet is not fixed.

5.1.2. Probe Placement for Binaries. A program is intrinsically composed of three parts: the *.bss* segment, the *.data* segment, and the *.text* segment [30]. As shown in Figure 6,

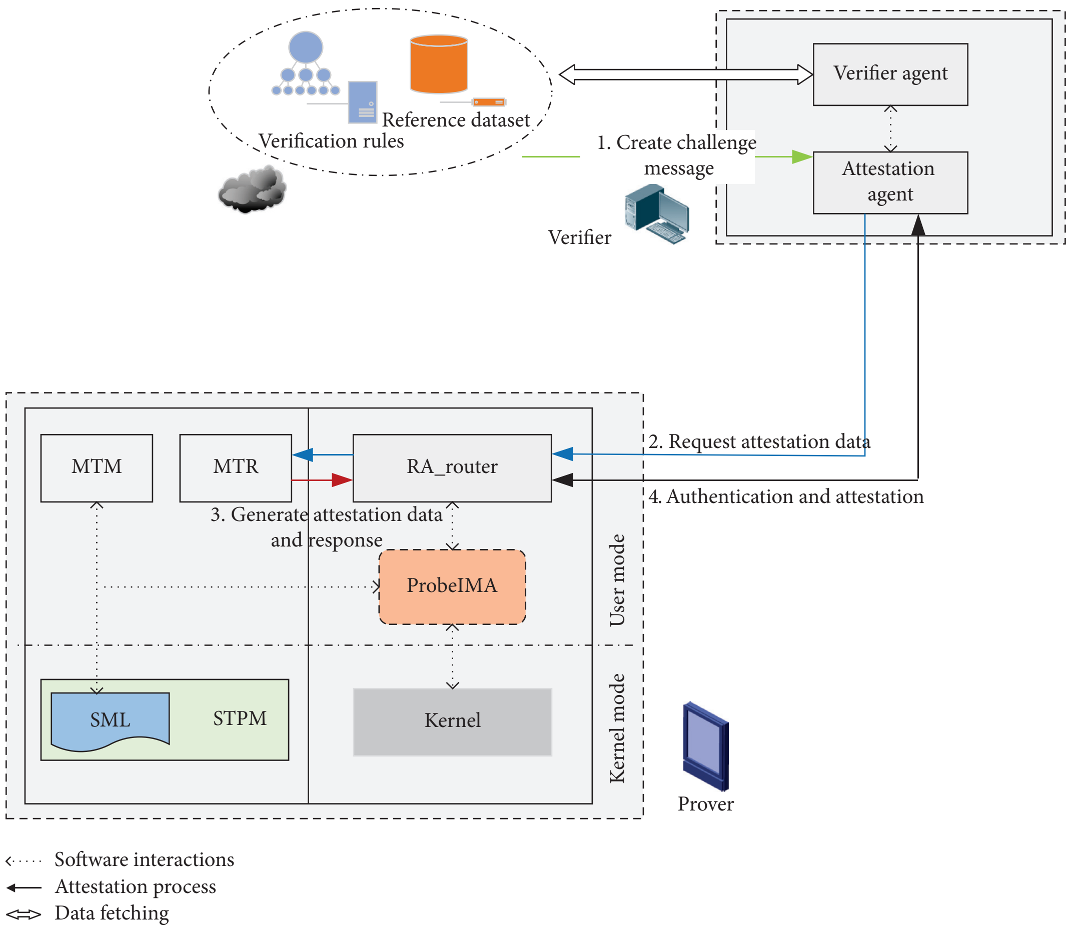
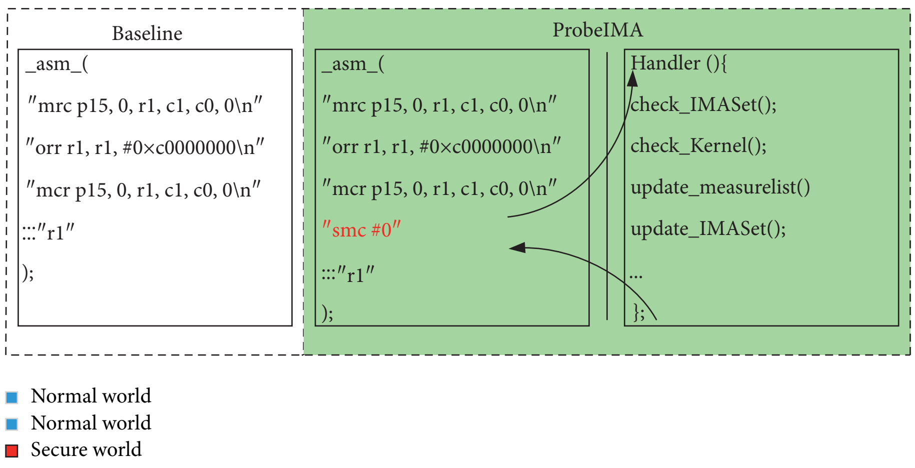


Figure 4: The TZ-MRAS remote attestation protocol.



Baseline:

```
_asm_(
"mrc p15, 0, r1, c1, c0, 0\n"
"orr r1, r1, #0×c0000000\n"
"mcr p15, 0, r1, c1, c0, 0\n"
:::"r1"
);
```

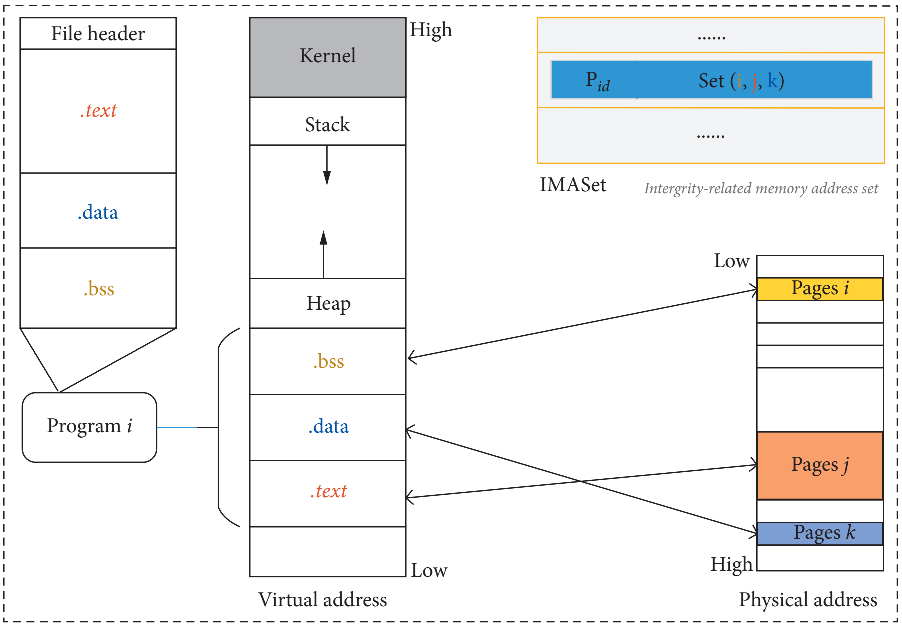
ProbeIMA:

```
_asm_(
"mrc p15, 0, r1, c1, c0, 0\n"
"orr r1, r1, #0×c0000000\n"
"mcr p15, 0, r1, c1, c0, 0\n"
"smc #0"
:::"r1"
);
```

```
Handler (){
check_IMASet();
check_Kernel();
update_measurelist()
update_IMASet();
...
};
```

Figure 5: The placement and triggering mechanism of the probe.

the *.text* and *.data* sections are generally solidified in the executable file loaded by the kernel from the binary. The *.bss* segment is not in the binary and generated by the process of initialization. The integrity of *.text*, *.data*, and *.bss* is equivalent to the integrity of the program itself. In Linux, the kernel creates a separate address space for each process, creating a new set of page tables. Page tables, a.k.a. memory translation tables, are stored in the kernel's memory fragment, and the update of the translation tables is also a normal memory write.It means that the secure world can obtain an unforgeable view of the normal world's translation tables, and dynamically maintain the IMASet is possible, which manages different process's page tables. On the other hand, due to the ARM processor can support two page tables



▨ Kernel space

□ User space

Figure 6: The design of the IMASet.

simultaneously, TTBR0 and TTBR1. The same virtual address bits may be mapped to different physical addresses. Therefore, we use the physical memory address to build the IMASet.

To capture the integrity changes of the application running in the normal world, the ProbeIMA inserts the SMC instruction at all program memory write instructions. When a write instruction of the memory page table included in the IMASet is detected, the IMA component of the secure world can determine the program ID corresponding to the target physical memory address and dynamically update the integrity record of the program. On the one hand, since ARM has fixed-length instructions, it is easy to find such instructions in the kernel binaries. On the other hand, since the normal world is accessible to the secure world, the secure world can read the key information, for example, the page table base address, directly. So, the components in the secure world can easily maintain the association information of the page table and the CA. The secure world obtains the program ID and identifies whether editing of the corresponding page table can cause the integrity change.

Because the measurement object of the integrity verification is not continuously distributed and stored in memory, the traditional hardware-based and software-based memory extraction methods extract the entire memory and extract the message from it, which has a higher performance and time cost. We gather all the page tables' set, which affects the integrity of the programs and kernels and only needs to extract data from memory according to the IMASet. It conforms to the law that the measurement objects are discrete distribution, which dramatically accelerates the extraction of measurement objects. Because the integrity measurement process of static measurement objects such as code, data, interrupt vector table, and system call table is relatively simple, we will not detail here.

In summary, we directly modify the kernel and CA's binaries to place probes at the instructions which can modify the key coprocessor registers and memory page tables. When the probe is triggered, the SMC instruction is executed and switches the execution environment to the secure world. The secure world analyses the trigger instruction and performs integrity check for the instruction of the valid trigger.

5.2. The Locality Principle-Based SML Construction Algorithm

5.2.1. Locality Principle. Locality principle-based SML is a tree data structure used to store the entity's integrity measurements. In LPSML, the root node is stored in the hardware-protected register PCR; the leaf nodes and intermediate nodes are protected by the trusted execution environment. Therefore, any nonphysical attack, including malware and Trojan virus, cannot be considered to threaten the security of SML in LPSML. At the same time, the memory space of the secure world is used as a buffer pool to buffer the application requests and avoid the construction and update operation being too frequent, which will increase the scalability of the construction mechanism and improve the performance of the approach. The operations involved in LPSML mainly include the statistics on the recent access frequency, traversal and updating of hash trees, and revocation of nodes. The composition of nodes in LPSML can be represented by a six-tuple [31]:

$$\langle \text{id}, h\text{value}, \text{raf}, l\text{child}, r\text{child}, \text{parent}\rangle, \quad (5)$$

where id is the node ID, and each application has an unique node ID, $h$value refers to the measurements of the program, raf is the frequency of the recent accesses, and $l$child, $r$child, parent represent separately the left child node, the right child node, and the parent node. Each time the LPSML is updated, the root node of the tree needs to be re-expanded into PCR.

Definition 1. Recent access frequency (RAF) [31]:

Assume that, in the recent period of time t, there are a total of m different program requests for access, and the number of requests for access per program is r_i; then, the recent access frequency (RAF) can be defined as the following equation:

$$\mathrm{RAF} = \frac{r_i}{\sum_{i=1}^{m} r_i}. \tag{6}$$

According to the locality principle of program access, when a program recently requests access, the program has a high probability of requesting access again in a short period. Accordingly, an application that has higher RAF means it has a higher probability of requesting access again, which is formally described as follows: suppose each program s_1, s_2 has a recent access frequency of $\mathrm{RAF}_1, \mathrm{RAF}_2$ in a time interval t_i and a recent access frequency of $\mathrm{RAF}_1', \mathrm{RAF}_2'$ in the next time interval t_{i+1}, and $P(\cdot)$ represents the probability of occurrence of an event; then,

$$\begin{aligned}&\exists \varepsilon > 0, \text{s.t.}\\ &P\left(\left(\mathrm{RAF}_1 > \mathrm{RAF}_2 \wedge \mathrm{RAF}_1' > \mathrm{RAF}_2'\right) \vee \left(\mathrm{RAF}_1 \leq \mathrm{RAF}_2 \wedge \mathrm{RAF}_1' \leq \mathrm{RAF}_2'\right)\right) > \varepsilon.\end{aligned} \tag{7}$$

Based on the above assumptions, the core mechanism of LPSML is to make the attestation path shorter for the applications which have higher recent access frequency, thereby saving time consumption and improving verification efficiency.

5.2.2. The Construction Algorithm of LPSML. Left-full tree (LFT) is a binary tree in which all left subtrees are a full tree, satisfying the following properties: (1) the depth of the LFT left subtree must be $k-1$; (2) the depth of the left subtree is greater than or equal to the depth of its right subtree. A full tree means that, in a binary tree, each node has two child nodes except for the leaf node.

A hash tree that uses an LFT as a storage structure is called a left-full hash tree. The left-full hash tree is used as the storage structure of integrity measurement data in our design. First, we use the principle of locality to calculate the weight of each measured entity, that is, we use RAF to assign values to all leaf nodes. Second, we generate a left-full hash tree and ensure that the sum of the weighted path lengths is the shortest.

Specifically, we demonstrate this construction algorithm through a practical scenario: we observed all GUI software collections in a Linux production environment (Ubuntu 16) and used scripts to count the frequency of manual opening of each program over a longer period of time. We found that application access in this scenario follows strong temporal locality. Therefore, we counted and normalized the number of access of different applications and built a software access frequency model that satisfies the principle of the temporal locality as follows: {0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.075, 0.081, 0.784}.

We refer to the Huffman tree to construct a left-full tree that meets the sum of the weighted path lengths is the shortest, as shown in Figure 7. HuffMHT is a full binary tree in which the root node contains all the subnode information, and the sum of the weighted path lengths is the shortest. The advantage is, first, RAF the larger the value of the RAF, the shorter the attestation path. An attestation path of entity $s7$ is shown as the yellow nodes. Second and more important, the left-full tree can dynamically adjust the number of nodes, thus storing the measurement values in the form of a left-full tree which can greatly simplify node insertion and revocation operations.

6. Experiment

In this section, we evaluate our prototype performances from two aspects of function and performance. The function test is used to verify the effectiveness of ProbeIMA's dynamic integrity measurement, that is, whether the system can measure and check anomalies and attack behaviours. The performance test is used to verify the efficiency and cost of ProbeIMA and LPSML. First, we implemented the OP-TEE trusted execution environment [32] on the hardware platform. We used trustlets running in the secure world to simulate the core functions of the STPM, including the secure storage and IMA components. Second, we examined the world switch process generated by ProbeIMA, focusing on the overhead of using the probe mechanism. Third, we built a control experiment in the desktop environment and performed performance analysis on LPSML.

6.1. Experimental Environment. We cross-compile an open-source OP-TEE trusted execution environment in Ubuntu 16.04 desktop system and deploy it to the Raspberry Pi 3 Model B Rev1.2 development board, which has a processor speed of 1.2 GHz and a 1 GB memory. OP-TEE is a popular TEE solution that can implement an integrated trusted OS application scenario. The secure world runs a customized mini operating system optee_os, and the normal world runs a 64 bit Linux system. The TPM functionality is implemented using the OP-TEE trusted application (TA), which provides an interface that conforms to the global platform's internal API [33] specifications for encryption and secures storage and other operations. The TA of the measurement module implements operations for initializing the signature key pair and PCR,

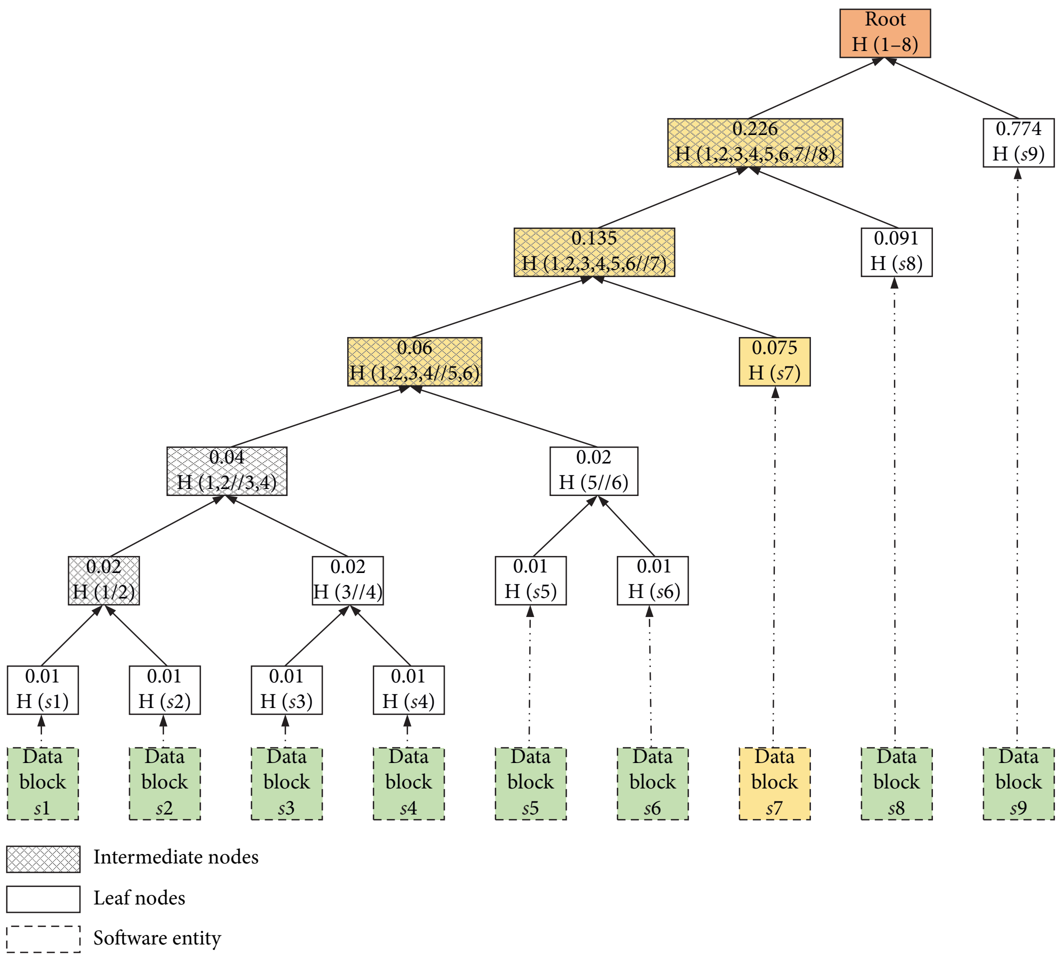


Figure 7: An example of our implementation of LPSML.

including computing and expanding the PCR value for simulating the TPM_quote operation. All of the security contexts are stored in the TrustZone secure RAM storage area.

A desktop experiment environment is built to perform the performance of the LPSML. The specific environment configuration is CPU @ 3.2 GHz, 4 GB memory, and the system is Ubuntu 15.04 LTS 64 bit.

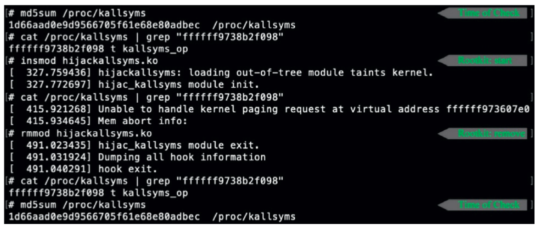
6.2. Security Analysis and Verification. As mentioned in Section 4, when booting up, if a system kernel has been authenticated to be trusted and had not been modified and all applications executing afterwards are trusted and had not been modified, then we consider that the current state of the system is trusted. Similarly, Arun et al. [34] used the dynamic hash integrity checker to perform integrity check for all history instructions at runtime to cope with the TOC-TOU threats. That is, the current integrity of a system that had not run a dangerous program or found an anomaly state of the kernel is considered to be trustworthy.

In fact, all possible attack behaviour must put the operation such as modify the system parameters or execute an illegal process as the iconic achievement. It means that, in the trusted computing field, all attack behaviours will result in changes in the system's integrity. Therefore, we put activities such as the malware activation and the configuration parameter modification as the security check trigger is reasonable and reliable.

An attack example is used to analyze the dynamic integrity detection capability of the ProbeIMA. Specifically, we created a simple rootkit to hijack the kernel function of the system call table. First, we determine the address of the target function through kallsyms_lookup_name. Secondly, the rootkit dynamically creates a kernel hook, inserts a payload, and hides the module with the hot patching function aarch64_insn_patch_text provided by the system. Finally, by loading and removing of this rootkit, we simulated a typical TOC-TOU attack, and an attack and a recovery behaviour are performed between the twice integrity checks.

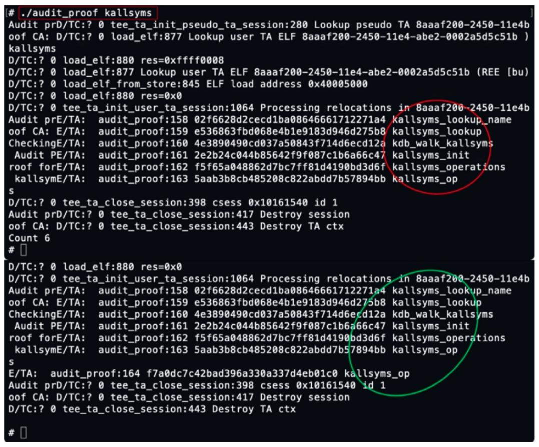
As shown in Figure 8, we measured the integrity of the kernel symbol table /proc/kallsyms before the execution and removal of the rootkit program, and both of them can correctly output their hash values. However, when the rootkit is running, by hooking the symbol memory address of kallsyms_op, the kernel symbol information cannot be print correctly.

Finally, we implemented a query component for integrity measurements in the secure world and provided a query interface to the normal world. The results are shown in Figure 9; although the hash value of the relevant kernel symbol has not changed between the two queries, we can still find that the



```
# md5sum /proc/kallsyms
1d66aad0e9d9566705f61e68e80adbec  /proc/kallsyms
# cat /proc/kallsyms | grep "ffffff9738b2f098"
ffffff9738b2f098 t kallsyms_op
# insmod hijackallsyms.ko
[  327.759436] hijackallsyms: loading out-of-tree module taints kernel.
[  327.772697] hijac_kallsyms module init.
# cat /proc/kallsyms | grep "ffffff9738b2f098"
[  415.921268] Unable to handle kernel paging request at virtual address ffffff973607e0
[  415.934645] Mem abort info:
# rmmod hijackallsyms.ko
[  491.023435] hijac_kallsyms module exit.
[  491.031924] Dumping all hook information
[  491.040291] hook exit.
# cat /proc/kallsyms | grep "ffffff9738b2f098"
ffffff9738b2f098 t kallsyms_op
# md5sum /proc/kallsyms
1d66aad0e9d9566705f61e68e80adbec  /proc/kallsyms
```

Figure 8: An example of rootkit loading and removal.



```
# ./audit_proof kallsyms
Audit prD/TC:? 0 tee_ta_init_pseudo_ta_session:280 Lookup pseudo TA 8aaaf200-2450-11e4b
oof CA: D/TC:? 0 load_elf:877 Lookup user TA ELF 8aaaf200-2450-11e4-abe2-0002a5d5c51b )
kallsyms
D/TC:? 0 load_elf:880 res=0xffff0008
D/TC:? 0 load_elf:877 Lookup user TA ELF 8aaaf200-2450-11e4-abe2-0002a5d5c51b (REE [bu)
D/TC:? 0 load_elf_from_store:845 ELF load address 0x40005000
D/TC:? 0 load_elf:880 res=0x0
D/TC:? 0 tee_ta_init_user_ta_session:1064 Processing relocations in 8aaaf200-2450-11e4b
Audit prE/TA:  audit_proof:158 02f6628d2cecd1ba08646661712271a4 kallsyms_lookup_name
oof CA: E/TA:  audit_proof:159 e536863fbd068e4b1e9183d946d275b8 kallsyms_lookup
CheckingE/TA:  audit_proof:160 4e3890490cd037a50843f714d6ecd12a kdb_walk_kallsyms
 Audit PE/TA:  audit_proof:161 2e2b24c044b85642f9f087c1b6a66c47 kallsyms_init
roof forE/TA:  audit_proof:162 f5f65a048862d7bc7ff81d4190bd3d6f kallsyms_operations
 kallsymE/TA:  audit_proof:163 5aab3b8cb485208c822abdd7b57894bb kallsyms_op
s
D/TC:? 0 tee_ta_close_session:398 csess 0x10161540 id 1
Audit prD/TC:? 0 tee_ta_close_session:417 Destroy session
oof CA: D/TC:? 0 tee_ta_close_session:443 Destroy TA ctx
Count 6
# 
```

```
D/TC:? 0 load_elf:880 res=0x0
D/TC:? 0 tee_ta_init_user_ta_session:1064 Processing relocations in 8aaaf200-2450-11e4b
Audit prE/TA:  audit_proof:158 02f6628d2cecd1ba08646661712271a4 kallsyms_lookup_name
oof CA: E/TA:  audit_proof:159 e536863fbd068e4b1e9183d946d275b8 kallsyms_lookup
CheckingE/TA:  audit_proof:160 4e3890490cd037a50843f714d6ecd12a kdb_walk_kallsyms
 Audit PE/TA:  audit_proof:161 2e2b24c044b85642f9f087c1b6a66c47 kallsyms_init
roof forE/TA:  audit_proof:162 f5f65a048862d7bc7ff81d4190bd3d6f kallsyms_operations
 kallsymE/TA:  audit_proof:163 5aab3b8cb485208c822abdd7b57894bb kallsyms_op
s
E/TA:  audit_proof:164 f7a0dc7c42bad396a330a337d4eb01c0 kallsyms_op
Audit prD/TC:? 0 tee_ta_close_session:398 csess 0x10161540 id 1
oof CA: D/TC:? 0 tee_ta_close_session:417 Destroy session
D/TC:? 0 tee_ta_close_session:443 Destroy TA ctx

# 
```

Figure 9: Comparison of integrity records at different moments.

number of queried hash value records has increased. Thus, the root node value stored in the PCR should be changed and also the result of the integrity proof. Therefore, ProbeIMA can effectively achieve runtime kernel integrity update.

6.3. Performance Cost of ProbeIMA.

6.3. Performance Cost of ProbeIMA. In this section, we evaluate the performance impact of ProbeIMA on the REE. We implemented a limited model for performance evaluation in the experimental environment. The main job is the design of the handling agent in the MonitorMode and the measurement TA in the TEE. Since the binary of the MonitorMode is packaged with the binary of the secure operating system in the OP-TEE project, the MonitorMode binary and trusted OS cannot be loaded separately; thus, we cannot measure the time consumption of the InvalidTrigger specially. Therefore, in the experiment design, we first counted the number of probes placed, and then we estimated the time overhead caused by different trigger modes according to the proportion of the probe in different stages.

$$\text{Cost}_{\text{ProbeIMA}} = \text{Times}_{\text{PT}} * \text{Cost}_{\text{PT}} + \text{Times}_{\text{NT}} * \text{Cost}_{\text{NT}}. \tag{8}$$

The total time consumption of the ProbeIMA is shown as equation (8), where Times_{PT} and Times_{NT} indicate the number of the valid triggers and the invalid triggers that triggered cumulatively while the system is running, Cost_{PT} and Cost_{NT} indicate the time delay generated by each.

The statistics for the prototype system are shown in Table 1; we built two user programs, RA_Router and

Table 1: Statistics of ProbeIMA.

(i) The number of probes placed in the core process			
Items	Kernel	RA_Router	Test_App_CA
Count of probes	207.1 k	2.19 k	0.03 k
(ii) The distribution of instructions in a valid trigger			
Phase	Binary load	World switch	Handler process
Count of instructions	0.15 k	1.2 k	10.2 k

Test_App_CA, in the untrusted operating system of the normal world, where RA_Router is used to forward the interaction data between the report module in the secure world and the remote verifier, including receiving and forwarding the attestation challenges and measurement data. Test_App_CA is part of the sample test program that runs in the normal world. Our experimental model did not model the exact cycle count but instead tracked the instruction count. To quantify the performance overhead caused by the proposed solution, we measured the time consumption generated by the operation of the system kernel and core program in the experimental environment. Furthermore, a controlled experiment baseline was setup to compare with our prototype. In baseline, we did not place the probe and only installed the secure world's IMA component.

We recorded the time consumption of the system kernel, RA_Router, and Test_App_CA running in two sets of environments (Table 2). A delay of 0.35 seconds was added during the system startup phase. Moreover, the time consumption of the startup and the execution of the user program are minimal, and the actual needs of the mobile system platform are met. Finally, according to the statistical results, the estimated average delay of a single invalid trigger is about 47 μs, while the average delay of a single valid trigger is 3.2 ms.

6.4. Performance Analysis of the LPSML. To evaluate the verification efficiency of our scheme, we created a simple attestation model with two entities in the desktop environment that are a user application as the attestation requester and a user application as the attestation responder. The responder maintains a collection of measurement objects in different sizes. A Monte Carlo method is used to generate attestation request sequences of the requester application. As described in Section 5.2.1, a shorter length of the attestation path indicates a more efficient verification efficiency. As a control experiment, we implemented a remote authentication scheme based on the balanced binary tree in the same hardware and software environment. We recorded the results of the average length of certification path $\text{length}_{\text{path}}$ under different *iol*, *m*, and *n*. iol means the strength of the local principle, *m* refers to the number of total nodes, and *n* denotes the count of update nodes each time. The results are shown in Figure 10.

The relationship between the attestation path length $\text{length}_{\text{path}}$ and the total number of nodes *m* is shown in

Table 2: The time consumption example (in seconds).

Schemes	Baseline		ProbeIMA		Extra cost	
	Start	Resume	Start	Resume	Start	Resume
Kernel	3.97	2.71	4.32	2.72	0.35	0.01
RA_Router	0.32	—	0.32	—	—	—
Test_App_CA	0.02	—	0.02	—	—	—

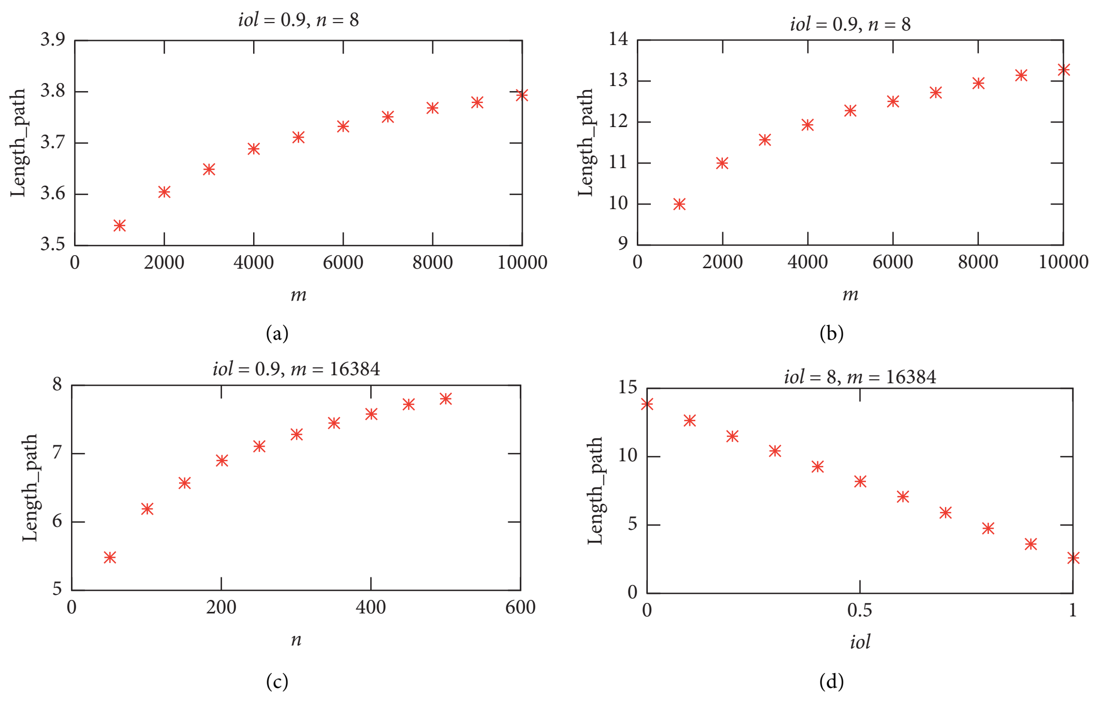


Figure 10: The diagram of verification efficiency.

Figures 10(a) and 10(b). The former presents that, as the total number of request nodes increases from 1,000 to 10,000, the average attestation path length varies from 3.5 to 3.9 in our scheme. In the later one, as a comparison experiment, another remote authentication scheme based on a balanced binary tree (BBT) in the same hardware and software environment is shown. The average attestation path length in the balanced binary tree scheme is equal to the depth of the tree, which increases from 10 to 14. Therefore, the LPSML made the average attestation path length shorten by about 69.63%.

Another advantage of LPSML is reducing the average path length of the modified value nodes, which can shorten the time consumption of recalculating the upper node hash values. We set *iol* equals 0.9 of (0, 1). Figure 10(c) shows that, as the number of updated nodes increases from 40 to 600 at the same time, the average updated path depth rises from 5 to 8 wich is far more shorten than the depth value 14 in the BBT. At last, we assume *n* at a typical value which is 8. With the strength of local principle *iol* increasing, the efficiency of the remote attestation is becoming better and better, which is shown in Figure 10(d).

From the above analysis, we can see that when the program requests access, the attestation path length of LPSML is obviously less than that of the balanced hash tree. When the intensity of locality is higher, the performance is better, which reduces the operational load of the server. As a result, the verification efficiency is improved.

7. Conclusion

A large number of mobile devices use the ARM-based application-specific integrated circuit, and it is necessary to study the remote attestation scheme under the ARM architecture. To build an efficient and economical remote attestation framework, we implemented the trusted remote attestation components by the TrustZone's highest privilege. First, we used TA to simulate the trusted attestation core modules, and all of the security-critical operations were executing in the TEE to achieve outstanding performance. Second, a probe-based dynamic measurement mechanism, ProbeIMA, is proposed to dynamically detect unknown fingerprints that appear in the kernel and process during operation. The ProbeIMA can effectively defend against TOC-TOU attacks and meet expected security needs. Third, according to the characteristics of the improved dynamic measurement of ProbeIMA, an LPSML-based optimization algorithm is proposed, which has the advantages of shortening the attestation path's length and improving the efficiency of platform configuration integrity verification.

Compared with existing solutions, the proposed solution has better versatility and is more suitable for low-cost heterogeneous systems.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (general program) under Grant no. 61572253 and the Aviation Science Fund under Grant no. 2016ZC52030.

References

[1] ARM Holdings, "Annual reports," 2019, https://www.armlife.com.ng/armlife_uploads/2019/10/2018_Annual_Report_ARMLife_Plc.pdf.

[2] P. Sparks, *The Route to a Trillion Devices*, ARM, Cambridge, UK, 2017.

[3] J. Winter, "Trusted computing building blocks for embedded linux-based arm trustzone platforms," in *Proceedings of the 3rd ACM Workshop on Scalable Trusted Computing*, pp. 21–30, ACM, Fairfax, VA, USA, October 2008.

[4] "Adoption of trustonic security platforms passes 1 billion device milestone," Trustonic, 2019, https://www.trustonic.com/news/company/adoption-trustonic-security-platforms-passes-1-billion-device-milestone/.

[5] Z. Huanguo and W. Fan, "A behavior-based remote trust attestation model," *Wuhan University Journal of Natural Sciences*, vol. 11, no. 6, pp. 1819–1822, 2006.

[6] W. Arthur and D. Challener, *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*, Apress, New York, NY, USA, 2015.

[7] C. Shepherd, N. A. Raja, and K. Markantonakis, "Secure remote credential management with mutual attestation for constrained sensing platforms with tees," 2018, https://arxiv.org/pdf/1804.10707v1.pdf.

[8] G. Coker, J. Guttman, P. Loscocco et al., "Principles of remote attestation," *International Journal of Information Security*, vol. 10, no. 2, pp. 63–81, 2011.

[9] R. C. Merkle, "Protocols for public key cryptosystems," in *Proceedings of the 1980 IEEE Symposium on Security and Privacy*, p. 122, IEEE, Oakland, CA, USA, April 1980.

[10] H. Raj, S. Saroiu, A. Wolman, R. Aigner, and J. Cox, "fTPM: a software-only implementation of a TPM chip," in *Proceedings of the 25th USENIX Security Symposium (USENIX Security 16)*, pp. 841–856, Austin, TX, USA, August 2016.

[11] S. Thom, J. Cox, D. Linsley, M. Nystrom, and H. Raj, "Trustzone-based integrity measurements and verification using a software-based trusted platform module," US20160048678A1 US Patent, 2016.

[12] W. Feng, Y. Qin, S. Zhao, and D. Feng, "AAoT: lightweight attestation and authentication of low-resource things in IoT and CPS," *Computer Networks*, vol. 134, pp. 167–182, 2018.

[13] D. Fu, X. Peng, and Y. Yang, "Unbalanced tree-formed verification data for trusted platforms," *Security and Communication Networks*, vol. 9, no. 7, pp. 622–633, 2016.

[14] Y. Zhang, L. Wang, Y. You, and L. Yi, "A remote-attestation-based extended hash algorithm for privacy protection," in *Proceedings of the 2017 International Conference on Computer Network, Electronic and Automation (ICCNEA)*, pp. 254–257, IEEE, Xi'an, China, September 2017.

[15] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "Trusted execution environments on mobile devices," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, pp. 1497-1498, ACM, Berlin, Germany, November 2013.

[16] "TPM 2.0 mobile reference architecture specification," Trusted Computing Group, 2019.

[17] Trusted Computing Group, "TPM mobile with trusted execution environment for comprehensive mobile device security," June 2012, https://trustedcomputinggroup.org/wp-content/uploads/TPM-MOBILE-withTrusted-Execution-Environment-for-Comprehensive-Mobile-Device-Security.pdf.

[18] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and implementation of a TCG-based integrity measurement architecture," in *Proceedings of the 13th USENIX Security Symposium*, pp. 223–238, San Diego, CA, USA, August 2004.

[19] Samsung Knox, "Secured by Knox," 2020.

[20] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proceedings of the NDSS*, pp. 191–206, San Diego, CA, USA, 2003.

[21] L. Nick, J. Petroni, T. Fraser, J. Molina, and W. A. Arbaugh, "Copilot—a coprocessor-based kernel runtime integrity monitor," in *Proceedings of the USENIX Security Symposium*, pp. 179–194, San Diego, CA, USA, August 2004.

[22] A. M. Azab, P. Ning, E. C. Sezer, and X. Zhang, "HIMA: a hypervisor-based integrity measurement agent," in *Proceedings of the 2009 Annual Computer Security Applications Conference*, pp. 461–470, IEEE, Honolulu, HI, USA, December 2009.

[23] B. Stelte, R. Koch, and M. Ullmann, "Towards integrity measurement in virtualized environments—a hypervisor based sensory integrity measurement architecture (SIMA)," in *Proceedings of the 2010 IEEE International Conference on Technologies for Homeland Security (HST)*, pp. 106–112, IEEE, Waltham, MA, USA, November 2010.

[24] A. M. Azab, P. Ning, J. Shah et al., "Hypervision across worlds: real-time kernel protection from the ARM trustzone secure world," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 90–102, ACM, Scottsdale, AZ, USA, November 2014.

[25] X. Ge, H. Vijayakumar, and T. Jaeger, "Sprobes: enforcing kernel code integrity on the trustzone architecture," in *Proceedings of the Third Workshop on Mobile Security Technologies (MoST) 2014*, San Jose, CA, USA, May 2014.

[26] Z.-Y. Xu, Y.-P. He, and L.-L. Deng, "Efficient remote attestation mechanism with privacy protection," *Journal of Software*, vol. 22, no. 2, pp. 339–352, 2011.

[27] Y. Zhu, L. I. Qingbao, C. Zhong et al., "Non-balanced binary hash-tree model for fine-grained integrity measurement," *Journal of Chinese Computer Systems*, vol. 35, no. 7, pp. 1604–1609, 2014.

[28] S. Pinto and N. Santos, "Demystifying arm TrustZone," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–36, 2019.

[29] J. D. Peter, "The locality principle," in *Communication Networks And Computer Systems: A Tribute to Professor Erol Gelenbe*, pp. 43–67, World Scientific, Singapore, 2006.

[30] J. Xu, Z. Kalbarczyk, and R. K. Iyer, "Transparent runtime randomization for security," in *Proceedings of the 22nd International Symposium on Reliable Distributed Systems*, pp. 260–269, IEEE, Florence, Italy, October 2003.
[31] Z. Wang, Y. Zhuang, and Q. Xia, "Mutual authentication-based ra scheme for embedded systems," *IET Information Security*, vol. 14, no. 2, pp. 232–240, 2020.
[32] Open Portable Trusted Execution Environment (OP-TEE), June 2020.
[33] "Trusted user interface API v1.0," GPD_SPE_020—GlobalPlatform, May 2019.
[34] K. Arun, Z. Mohamed, and K. Ramesh, "Architecture support for dynamic integrity checking," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 321–332, 2011.