Enabling Security-Enhanced Attestation With Intel SGX for Remote Terminal and IoT

Juan Wang, *Member, IEEE*, Zhi Hong, Yuhan Zhang, and Yier Jin, *Member, IEEE*

***Abstract*—Along with the advent and popularity of cloud computing, Internet of Things, and bring your own device, the trust requirement for terminal devices has increased significantly. An untrusted terminal, a terminal that runs in an untrustworthy execution environment, may cause serious security issues for enterprise networks. With the release of Software Guard Extension, Intel has provided a promising way to construct trusted terminals and services. Utilizing this technology, we propose a security-enhanced attestation for remote terminals, which can achieve shielded execution for measurements and attestation programs. Furthermore, we present a policy-based measurement mechanism where sensitive data, including secret keys and policy details are concealed using the enclave-specific keys. We implement our attestation prototype on real platform with Intel Skylake processor. Evaluation results show that our attestation system can provide much stronger security guarantees, yet incurs small performance overhead.**

***Index Terms*—Attestation, Internet of Things (IoT), remote terminal (RT), secure enclaves, Software Guard Extension (SGX).**

I. Introduction

With the rapid development of cloud computing, Internet of Things (IoT) and bring your own device, we are witnesses to an explosive growth in the number of terminal devices [1], [2]. However, the security issues of terminal devices have become increasingly important. Terminal devices are vulnerable to various security threats. A large number of terminal devices have been hacked just due to simple password policy. As a result, a substantial amount of efforts are required to measure the terminal devices and attest them to be trusted [3]–[6].

Nowadays there are mainly two methods for constructing and verifying the trust of terminal devices. The first method is based on trusted platform modules (TPMs). Under this method, signature and secret keys related to the boot process are stored inside the TPM platform configuration registers [7], [8]. While the measurement value is signed by TPM and sent to a prover, the prover will verify the signature and the integrity of measurement results to attest the trust of attestees. The other method leverages TrustZone to construct trusted terminal [9], [10]. Trusted services are isolated in a secure world while untrusted services are running in a normal world. A trusted service is loaded into a trusted domain after the secure boot process has finished. The untrusted services can call the trusted service through the trusted application program interface (API). Randomly generated keys can be obtained from TPMs, or from other systems such as International Mobile Equipment Identity or physical unclonable functions [11]. These keys are used to sign and encrypt a measurement value.

However, the two methods above still have shortcomings and challenges. The TPM-based method can only securely protect keys, measurement values, and other sensitive data due to design and performance. TPMs cannot ensure the runtime isolation of program code and data. TrustZone-based systems suffer from the lack of built-in authentication when normal-world software communicates with the secure world. In addition, the current method does not consider the security of the attention program itself [12], such as the isolation of attestation program, the security process of verification procedure, and the protection of transmission module.

Aiming at these challenges, we propose a security-enhanced attestation for remote terminals (RTs) and IoT devices. Our method can achieve shielded execution for measurements and attestation programs. Furthermore, it can measure RT based on the custom policy and also the sensitive data including keys. The policy is sealed by the enclave-specific keys. We implement our prototype on platform using Intel Skylake processor and evaluate its performance. The results show that the overhead of the measurement module just increase about 3% and the cost of attestation module excluding measurement module is lower than open source attestation platform, e.g., OpenAttestation (OAT) [13].

Our contributions can be summarized as follows.

1) We propose a security-enhanced remote attestation method for RTs and IoT devices. Our method not only has the small trusted computing base but also constructs dynamic attestation method based on multiple enclaves. Meanwhile our method achieves the sealed storage of keys, isolated running of attestation program and secure

Manuscript received September 21, 2016; revised May 24, 2017; accepted August 13, 2017. Date of publication September 7, 2017; date of current version December 20, 2017. This work was supported in part by the National Natural Science Foundation of China under Grant 61402342, Grant 61173138, and Grant 61103628, and by the National Basic Research Program of China (973 Program) under Grant 2014CB340600. This paper was recommended by Associate Editor M. Huebner. *(Corresponding author: Yier Jin.)*

J. Wang, Z. Hong, and Y. Zhang are with the Department of Computer Science, Wuhan University, Wuhan 430072, China, and also with the Key Laboratory of Aerospace Information Security and Trust Computing, Ministry of Education, Wuhan 430072, China (e-mail: jwang@whu.edu.cn; whuhongzhi@foxmail.com; yh_zhang@whu.edu.cn).

Y. Jin is with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611-6200 USA (e-mail: yier.jin@ece.ufl.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2017.2750067

communication channel. A challenger can attest RT trust in a secure environment.
2) We present policy-based measurement mechanism. Administrators can collect and monitor the runtime status through the custom measurement policy, while the policy will be protected by the Software Guard Extension (SGX) seal key.
3) We implement RT attestation service for the first time on real physical platform with Intel Skylake processor running Linux. Furthermore, the performance is evaluated and the results show that the additional overhead is trivial.

The remainder of this paper is organized as follows. Section II provides some background information relating to our system including problem description, the threat model and Intel Software Guard Extension. Section III provides a system overview. Section IV introduces the design of secure isolation. Section V describes our policy-based measurement mechanism. Section VI shows the security-enhanced attestation procure. Section VII describes the implementation and evaluation of our prototype and presents the results and analysis. Section VIII shows some related work. Section IX concludes this paper.

II. Background

A. Problem Description

The trust of terminal devices is critical to enterprise networks because they are vast and vulnerable to attacks. The remote attestation mechanism of TCG can report the environment fingerprint of devices to attesters so as to validate the identity and the trust status of attestees. However, TPM-based attestation cannot protect the attestation program from tamping with and leaking sensitive information in runtime execution. ARM TrustZone can also be used to implement attestation of devices, but the transfer process from secure world to normal world and the trusted API of TrustZone service are vulnerable to attacks. Furthermore, attestation program protected by TrustZone may be threatened if one of programs in secure world has security weaknesses since all of protected programs run in the same isolated memory region.

Aiming at the challenges above, we propose SGX-based attestation service that can achieve security-enhanced attestation service. Our approach separates attestation service into untrusted module and trusted module. Meanwhile we isolates trusted attestation modules to different enclaves so that protecting the security of attestation program. We also present policy-based measurement. Challengers can send its measurement policy to attested entities. Attestation program will collect the measurement values of remote devices according to the measurement policy, sign and send them to the challengers. After validating the identity of the measurement values, the attesters will compare the measurement result with the baseline value which is collected in a clean state of the attested platform. If the measurement value matches the baseline value, we can determine that the attested platform is trusted otherwise it is untrusted. In the attestation procedure, we uses SGX quote operation to implement the signature of measurement values.

B. Threat Model

We follow the assumptions in the SGX model, where the processor is trusted and has not been tampered with. We allow the adversary to carry physical attacks on the system, such as modifying memory and changing I/O signals. We also assume that enclave programs are trusted and other programs in the system, including the BIOS, OS, and VMM are untrusted. We also assume the certificate authority (CA) to be trusted. However, adversaries can control the system outside the enclave, such as inserting malicious software in the system, and changing configuration parameters. In addition, the adversary may monitor and control network communication and will further attempt to impersonate the RT as to attempt to defeat server-side validation.

Note that we do not consider distributed denial of service attacks and side-channel attacks such as timing attacks, cache-collision attacks because that type of attacks can be mitigated by current defense mechanisms [14]–[16]. In addition, other side channel attacks, such as power analysis, require hardware modifications, and are ultimately a limitation of our approach.

C. SGX

The SGX by Intel Corporation was announced in Q1 2013 as an extension to the x86_64 instruction set architecture. It provides a way for applications to secure a portion of its address space and place data and code within this container. Intel calls this container an *enclave*. There is no set limit to the number of *enclaves* a process can own. Furthermore, it is important to notice that a process is strictly limited by the hardware in directly accessing the enclave's contents, whilst the enclave is free to access any portion of the process's memory.

SGX assumes that everything on a system with the exception of the processor can be compromised. That is, OS, drivers, BIOS, hypervisor, and system management mode are untrustworthy. As such, any secrets in an enclave remain protected even when the attacker has full control of the computer system. Code and data within the enclave is encrypted and cryptographically signed as to ensure its secrecy and integrity. The boundary of the security mechanism is the CPU package itself. Enclave code and data inside the CPU remain unencrypted. If this code or data is ever to leave the CPU package, it is encrypted and cryptographically signed. If a portion of an enclave is to enter the CPU again, it is decrypted inside the CPU after the integrity checking. This mechanism prevents bus sniffing, tampering with memory and cold boot attacks against an SGX-enabled system.

1) Enclaves: Enclave is a protected container for sensitive data and code [17]. SGX allows applications to be specified the trusted part and untrusted part. The code and data sections of trusted part need isolated protection. It is not necessary to do extra works on data or code before creating an enclave, but the data and code must be measured when loading into an enclave [18]. Once the protected part of an application has been loaded into an enclave, SGX protects them from external software, no matter it is a malicious program or just a normal one.

Data in memory is vulnerable to probing and malicious tampering. For example, memory leak attacks can potentially leak sensitive data from the system. Although solutions such as full memory encryption [19] or oblivious memory [20] have been proposed, these systems are not deployable in practice without considerable change. Furthermore, not all portions of memory contain sensitive information. Intel SGX opts by providing a special memory location, called the enclave page cache (EPC), where code and data from the enclaves is kept [21]–[23]. SGX also defines a security boundary on the CPU package itself. Any enclave-related data or code leaving the CPU is encrypted by a memory encryption engine (MEE) and kept in the EPC. When enclave code or data enters the CPU, the MEE decrypts it. Consequently, an attacker can only obtain ciphertext by leaking enclave code or data from outside the enclave.

Code outside an enclave has no access to the data inside, and code inside an enclave can only access the data and code belongs to the same enclave. For the memory space outside the EPC, the memory access mechanism has no difference with the normal. Such a memory protection mechanism, not only prevents the data inside an enclave from being tapped or tampered by malicious software, but also forbids the code inside an enclave to get data from other enclaves.

When a process terminates, its enclave instances are destroyed and the data and code it holds will disappear. To preserve some secret data in an enclave for future use, SGX offers a sealing function. Sealing can encrypt the data inside an enclave and store them on a permanent medium such as a hard disk drive, so the data can be used the next time. When sealing data, there are two options available: sealing to the current enclave using the current version of the enclave measurement (`MRENCLAVE`) or sealing to the enclave author uses the identity of the enclave author (`MRSIGNER`). In this paper, we use both mechanisms.

2) Attestation in SGX: In Intel SGX, attestation is the process of demonstrating that a piece of software has been authenticated on the platform and is being protected by an enclave [21]. Once an enclave is loaded, it is safe for a third party to communicate sensitive data to it. To achieve this goal, platform needs to supply third party with a credential which reflects its enclave security information and enclave signature. SGX supports two kinds of attestation, local (intraplatform) attestation and remote (interplatform) attestation. Local attestation is designed for enclaves on the same platform. When the attestation process completes, a secure session is established between two enclaves and they can call function and get data from each other. Remote attestation is a mechanism designed for the attestation between an enclave and a remote (not in the same platform) party. A remote challenger can get enclave information and platform security state through this process, then it will decide whether the enclave and the platform are trustworthy according to its local security configuration.

In the process of local attestation, an enclave asks hardware to generate a credential, known as a report, and send this report to another enclave on the same platform which can verify this report. An enclave report contains the following data: measurement of the code and data in the enclave, a hash of the public key in the independent software vendor (ISV) certificate, user data, other security related state information, and a signature block over the above data.

For remote attestation, an application can also send an enclave report to a quoting enclave to produce a type of credential that reflects enclave and platform state. This credential is called quote which is signed with an EPID private key [22]. Quoting enclave is an Intel provided enclave, which can process enclave report and convert report into enclave quote. Only the quoting enclave has access to the Intel EPID key. The quote is a data structure used for remote attestation and its main content is the same with report.

This quote can be passed to entities on another platform for verification. A quote includes the following data: measurement of the code and data in the enclave, a hash of the public key in the ISV certificate, the product ID and security version number of the enclave, attributes of the enclave, user data and a signature block over the above data.

III. System Overview

Our main goal is to achieve the trusted remote attestation of RTs and IoT devices by using the SGX technology. The proposed solution can prevent against security threats during remote attestation and monitor the runtime status of the RT, so as to detect it in real time and eliminate the possible risks of tampering in the RT.

A. Design Overview

The idea of our system revolves around sensitive modules during remote attestation being isolated into enclaves. The sensitive modules include the measurement module, the key storage module, the verification module of the server side, and the session module. The enclave can provide isolation for these modules and sealing storage for keys so as to prevent attackers, including malicious insiders, from tampering with programs or stealing keys and other sensitive information. When the critical status or configurations of attestees have been modified, it will be detected immediately by the attestation server and then the attesters can judge whether the terminals are trusted. Administrators can also take relevant measures to control the terminal. For example, they can recover the tampered data or disable the terminal to access the high-security services and even temporarily disable access to the enterprise internal network.

As shown in Fig. 1, the system is composed of remote attestation server (RAS), RT, and CA. The RAS is the verifier for remote attestation and the RT is the prover to be verified. The hardware of each side should support SGX. To prevent malicious attackers from tampering the attestation process and eavesdropping attestation results, we put main modules of our system into secure enclaves, such as the transport layer security (TLS) module, the measurement module, the attestation service, and the verifying module. We also use the sealing storage of SGX to protect the critical keys and security policies in the process of attestation. The main process of the system is described as follows.

1) RAS and RT start the TLS service and attestation service. The services are then called into the respective

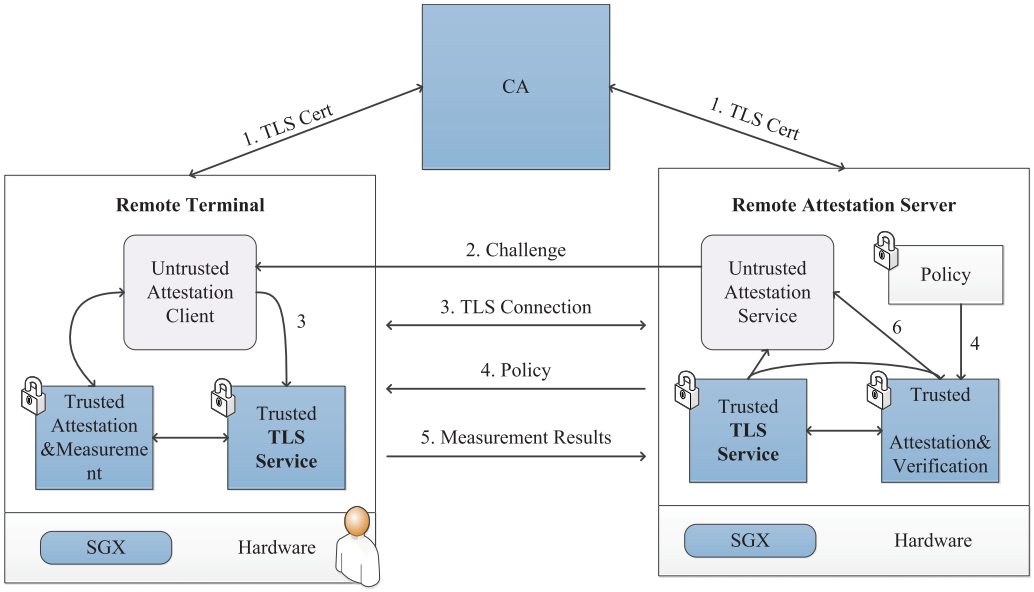

Fig. 1. Design framework of attestation.

enclave to be executed. TLS service and attestation service generate a private–public key pair and request the CA for issuing the public key certificates. The private key is sealed by the *Sealkey* which is generated in the corresponding enclave based on the public key of signer.

2) The attestation service of RAS challenges the RT for trusted verification.
3) The attestation service of RT calls the local TLS service to build the security session channel with RAS and then sends its attestation public key to RAS.
4) The attestation service of RAS sends the trusted verification policy to RT through the security channel. This policy is encrypted by the public key of RT.
5) In the enclave, RT decrypts the policy and calls the measurement program to detect the system according to the policy. RT quotes the measurement results and the list of measurement and then sends the results to RAS.
6) RAS verifies the signature and compares it with the baseline values stored in the server. If the result is the same as the baseline value, it verifies that the terminal is trusted.

B. Trust Policy Model

In our system, the trusted policy determines the information that needs to be verified in the remote attestation process. The trusted policy includes a policy of integrity checks, the system environment, and configuration information. The integrity policy defines which modules in the system need to be verified. The system environment includes the version of the system and kernel, the main information of devices, and a list of critical programs in the system and their version numbers. The configuration information includes password policies, service settings, and user settings. We obtain a baseline value for the system information on the first controlled run of the system by using the trusted policy to train the attestation model. Baseline values can also be obtained in a different way, such as detecting the system for this data using a specially crafted program. In our system, we provide a Web interface for administrators. Therefore, the trusted policy of terminal can be customized by the administrators and be sent to the terminal. The system also allows the administrators to modify the trusted policy.

IV. Enforcing Security Isolation

As we mentioned above, our remote attestation system is composed of an RT, an RAS, and a CA. In the following, we describe the key components of our system and analyze the detailed construction of each modules. To prevent malicious attackers tampering and to protect the communication, the security of the storage, and keys the key module of the RAS and RT should be isolated and sealed.

A. Remote Terminal

As shown in Fig. 2, RT is divided into four modules, including TLS service module, attestation module, measurement module, and untrusted modules.

TLS service module, being responsible for generating keys and obtaining certificates, is further divided into the keys part, the decryption and encryption part, and the network communication part. The keys part generates asymmetric key pairs for communicating with RAS, and communicates with CA to obtain the necessary public key certificates. Once the certificates have been obtained, the keys part of the module will exchange the certificates information and negotiate a session key with RAS by means of network part. This session key will be utilized by attestation module and measurement modules. In order to protect the integrity and confidentiality of private key and certificates, the keys part are hosted inside an SGX enclave. Encryption and decryption operations, which associated with the session key and performed by encryption and decryption part, are also hosted inside of an SGX enclave to

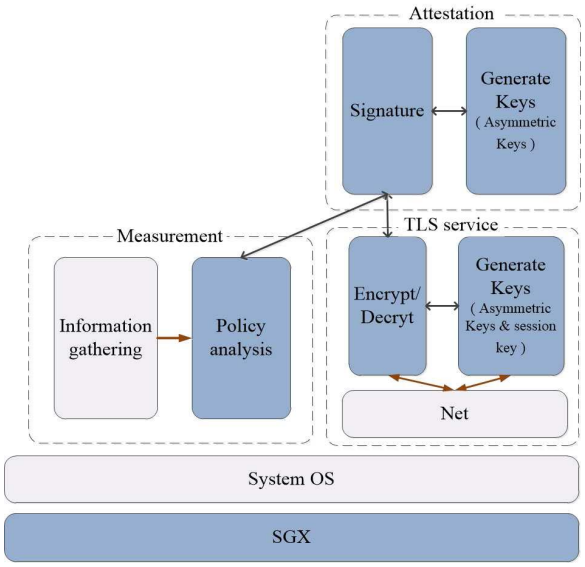

Fig. 2. Isolation design of RT.

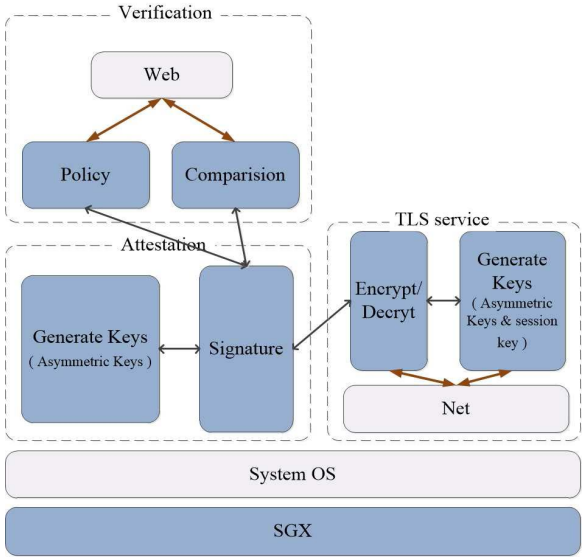

Fig. 3. Isolation design of RAS.

guarantee the confidentiality of the session keys. The untrusted network communication part is responsible for the transfer of data between RAS and CA.

The attestation module provides keys generation and signing functionality and is divided into two parts. Due to their nature in confidentiality, these parts are run inside enclaves. Different from the keys part of TLS service module, this one is used to generate a key pair to prove its identity in remote attestation process. During attestation process, signature part uses the private part of their key to sign the measurements. After receiving the policy file and data from RAS's attestation module, the signature part uses RAS's public key to verify its identity and then transfers it to other modules according to the process.

In measurement module, the policy analysis part parses the policy file and generates a checklist. To ensure confidentiality of the policy file, it runs inside an enclave. The message gathering part runs partially within enclave, while other parts running outside due to the invoked system calls. The policy file is sealed inside the enclave and stored on disk.

B. Remote Attestation Server

As shown in Fig. 3, RAS consists of four modules, including TLS service module, attestation module, verification module, and the untrusted module. Corresponding to RT's TLS service module, RAS's TLS service module are also hosted inside an SGX enclave after the system is booted. This module is responsible for generating RAS's key pair and communicating with CA to obtain the public key certificate. After obtaining the certificate, it will communicate with RT to exchange the certificate and the negotiated session key. For function independent consideration, attestation module can be divided into keys part and signature part. After receiving the measurement from the measurement module, the signature part uses the RT's public key to verify the signature of the measurement and transfers it to the verification module. Also, in attestation process, the signature part uses its own public key to sign the policy and other data to prove its identity.

The policy part and the comparison part, which are the key components of verification module, cooperate in measurement process. We designed the Web part for users to define the policy which will be translated as policy file by the policy part. After receiving the measurement from RT, the comparison part compares RT's measurement results to the local data and provides feedback to the Web part. All the policy file and measurement results should be sealed inside the enclave and stored in disk.

V. Measurement

The measurement module is used to collect system main properties according to the measurement policy. The policy will be sent to RT from attestation server. Once terminal has received measurement policy, it will launch measurement programs into an enclave to obtain the required system information. The measurement generally includes system information, security policy, running status, and system integrity policies.

System information consists of operating system version, kernel version, devices information, and installed applications. Security policies can be used to collect secure configuration about authentication, access control and security audit to measure the status of the system security policy. For example, for authentication, we gather and check user names as well as the policy of password validity and complexity. Also we can obtain the access control policy to check whether the system configuration meets required configuration values. In addition, the security audit policy can be collected to make sure the security audit is enabled and the audit policy is correct.

In order to monitor malicious behaviors, the measurement module is designed for detecting system's running status, such as running processes, opened services, and opened ports. Moreover, the integrity of the system boot will be measured. The integrity values of system boot can be sealed with SGX

key. However, currently enclaves can only be created and initialized in kernel mode, but they cannot be allowed to be entered in the kernel mode. Hence, we need to invoke the enter instruction in user mode and then return to the kernel mode.

VI. Security-Enhanced Attestation

The purpose of attestation is to collect measurement values of the RT and verify these values. When getting real-time measurement values, we compare them with the baseline values that we obtained in clean state and then decide whether they are trustworthy. Current remote attestation such as OAT, cannot guarantee its own security [13]. In order to solve this problem, we present a new remote attestation method combine with Intel SGX base on TLS. The main process of our remote attestation is depicted as follows.

1) *RT→RAS: Request connection*. RT initiates a connection request to RAS.
2) *RAS→RT: MSG1. MSG1 = cert*. RAS receives the RT's connection request and sends its own certificate to RT.
3) *RT→RAS: MSG2. MSG2 = pk1*. RT generates the ECC key pair pk1, priv1, and sends pk1 to RAS.
4) *RAS→RT: MSG3. MSG3 = pk2 || quotetype || policy || Sign(pk1,pk2) || MAC(pk2 || quotetype || policy || Sign(pk2,pk1))*. RAS generates the ECC key pair pk2, priv2. Choosing the type of quote and the measurement policy it needs, using the private key to generate signature of pk1 and pk2. By DH protocol and ECC nature, RAS can get sharedkey, and use sharedkey to generate message authentication code. Send all of the above information to RT.
5) *RT→RAS: MSG4 MSG4 = E(quote || measurement || hash(quote || measurement))*. RT gets MSG3, generates quote according to the quote type and generates measurement according policy. Then, RT generates the hash of quote and measurement. Finally, RT uses the sharedkey to encrypt the above and sends it to RAS.

Finally, RAS receives the encrypted result which will be decrypted and checked in an enclave. If the measurement result and quote fit in with the security requirement, RAS will validates the RT as a trusted terminal.

The advantage of this attestation mechanism is that we use the SGX to achieve the double-sided attestation based on enclave and SGX remote attestation. So both session and attestation process are securely protected.

VII. Implementation and Evaluation

A. Implementation Details

We implemented our method on a computer with an Intel Skylake processor i7 6700 and memory of 8 GB. The operating system we use is Ubuntu 14.04/64 bits and the compiler we use is g++ version 4.8.4.

We use the library `libsgx_urts` offered by SGX to implement enclave creating and destroying. SGX provides a tool called Edger8r, it can divide our program into trusted and untrusted parts according to the EDL file which is written by us. Furthermore, it generates the proxy port so that the trusted and untrusted part of the program can use enclave call (`ecall`) and out call (`ocall`) to communicate with each other.

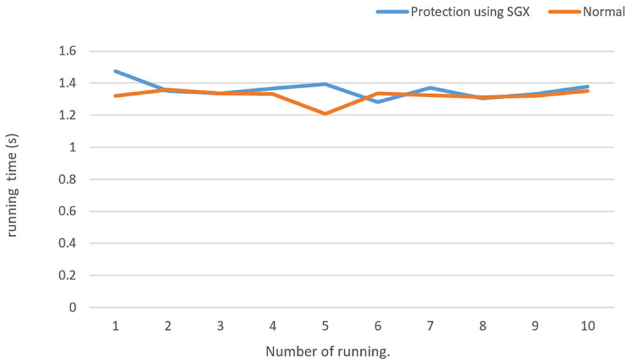

Fig. 4. Time cost between measurement program in enclave and outside.

SGX provides a cryptographic algorithm library `SGX_tcrypto`. It supplies some common algorithm such as SHA256, ECC256, and AES. The public key cryptographic algorithm we used in this paper is ECC256. We use `sgx_ecc256_create_key_pair()` to generate the key pair in enclave and we use the SHA256 as hash algorithm. As for the important random number nonce, we call `sgx_read_rand()` to produce it in enclave.

Our measurement modules includes 1785 lines C code. TLS and attestation modules include about 9600 lines Java code and 1000 lines C code.

B. Evaluation

1) Enclave Performance: Using SGX technology in the protection of program security may introduce additional overhead inevitably. Through analysis, we believe that extra overhead introduced by SGX mainly from two aspects.

1) Creating enclave, destroying enclave, and some other management operations on enclave.
2) `ecall` and `ocall` in a program may cause the program to switch its execution code between the trusted and untrusted parts.

We found that the structure of an SGX program may influence the running time at some degree in the process of actual operation. Therefore, we may optimize our program by designing an appropriate program structure, dividing a program into trusted and untrusted parts properly, avoiding frequent `ecall` and `ocall` to reduce the additional overhead introduced by SGX. We run our measure program in an enclave and outside enclave (normal) ten times to evaluate the influence on program performance caused by SGX. The results is depicted in Fig. 4.

Average running time for measurement program protected by SGX is 1358.83 and 1320.67 ms for the normal. From the experimental data, the average running time of our SGX enhanced measurement program is about 3% more than that running in the normal environment.

TABLE I
SEAL AND UNSEAL OPERATING TIME

Data Size	Seal (us)	unseal (us)
0.9kb	107.3	74.5
1.8kb	115.6	76.3
3.6kb	118.5	76.3
7.2kb	118.8	77.4

In addition, we also tested CPU utilization of measurement in enclave and in normal environment. We found that the program protected with enclave does not introduce overhead for CPU utilization. The CPU utilization of program in enclave and in normal environment is almost equal.

2) Seal Performance: We need to use sealing function offered by Intel SGX, so we should also consider the influence caused by seal and unseal operation. We perform seal and unseal operations on different size data including 0.9, 1.8, 3.6, and 7.2 kB. The average operating time they cost is listed in Table I.

3) Attestation Performance: In order to test the performance about the attestation described in Section V, we run our SGX enhanced attestation along with normal OAT for several times. We measure the time from an RT accepting the challenge to sending its measurement back to RAS. We want to illustrate that our design is feasible and efficient here and we believe that the measurement should be determined by the specific platform and, so the measurement policy we use is empty in our evaluation. The terminal which is attested by RAS will not perform measurement work and return an empty measurement. As shown in Fig. 5, the running time cost of the SGX enhanced attestation is lower than the original OAT, during the attestation process.

The main reason is that the original architecture of OAT uses TPM to save and quote measurement values and the longer time overhead of TPM's I/O operations increases the original OAT's time overhead. Compared with OAT, our security-enhanced attestation mechanism relies on CPU instructions to save and quote measurement values and does not have the interaction with I/O, hence its time overhead is lower that the traditional OAT.

C. Security Analysis

In the overall architecture, our design provides three security features as follows.

1) Providing runtime protection for attestation programs against runtime attacks. Therefore, our multiple modules are placed in different enclaves for protection. So when an attacker tries to perform tempering attacks, the attack is invalid for the integrity measurement of the enclave. At the same time, when an attacker tries to perform a memory leaking attack, the attacker only gets the encrypted memory and cannot get real data in protected memory.
2) The integrity protection of the key data. The generation, use, and storage of policy files and keys are in the protection of enclave in running time. When these key data need to be stored on disk, they are stored after being encrypted and measured to prevent from being tampered. During the communication between RAS and RS, these key data have been protected by TLS protocol and session key encryption. Attackers could not get the data through eavesdropping on communication channels.
3) Customized and trustworthy attestation content for remote attestation. Users can use policy files to customize their needs for attestation. And compared with the TPM remote attestation, our design is more focused on runtime security of RT. In the realization of the integrity measurement, we use the CPU-related hardware information which is physical unclonable and collect the system environment information. So it can effectively prevent counterfeit attacks.

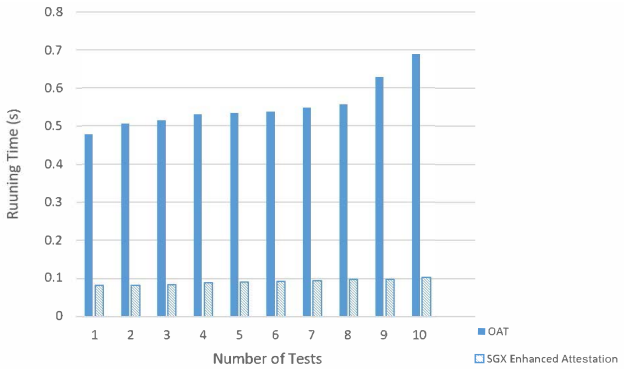

Fig. 5. Time cost between security-enhanced attestation and OAT.

VIII. RELATED WORK

Validating the running state of devices is an important task for the security of RTs and IoT devices [3]–[6]. Remote attestation has been used to resolve this issue. That is, the terminal can prove to remote server what software is running on the devices and whether they have been tampering with.

TPM-based attestation leverages a security chip called as TPM [7], [8] to ensure that a system platform has been securely launched. For this method, TPM serves as the root of trust and provides the isolated storage for platform keys and measurement values. Attestation program collects and reports the measurement values which is signed and securely saved through TPM so as to validate the trust of attestees. Brickell *et al.* [24] and Camenisch *et al.* [25] also presented direct anonymous attestation protocol which can provide anonymous identity based on group signature. However, TPM-based attestation cannot achieve runtime protection for attestation program.

GlobalPlatform first announced their own standardization of the TEE in 2010 [26]. TEE leverages the isolation mechanism of ARM TrustZone to provide secure execution environment for mobile application [9], [10], [27], [28]. ARM TrustZone is also used to attest the running environment of devices. Santos *et al.* [29] presented an approach of building trust chains on mobile and construct the trusted language runtime that protects the confidentiality and integrity of .NET mobile application from OS security breaches. Li *et al.* [30] proposed a secure online mobile advertisement attestation using ARM

TrustZone. However, TrustZone-based attestation is still vulnerable to attacks because attestation programs share the same secure world with others program.

Our method leverages software guard extension to provide secure isolated environment for remote attestation program and data. Attestation can be implemented in multiple enclaves on real physical platform. It achieves the confidentiality and integrity for main code, data and transmission during the whole attestation procedure.

SGX [21], as a new hardware-aided security technology, has been used to construct trusted computation environment. VC3 [31] implements the verifiable and confidential execution of MapReduce jobs in untrusted cloud environments using SGX. The sensitive code and data during the MapReduce process are protected with isolation and sealing storage. Haven [32] achieves shielded execution of unmodified legacy applications, including SQL server and Apache, on a commodity OS (Windows) and commodity hardware. It builds on Drawbridge, a system supporting low-overhead sandboxing of Windows applications. SGX is used to isolate the Drawbridge into a sure enclave. Moreover, VC3 and Heaven have been implemented in SGX emulator. SCONE [33] uses SGX to protect container key processes from outside attacks that support a set of shields including file system shield, network shield and console shield. Meanwhile it also designs an asynchronous system call mechanism to reduce SGX-imposed enclave transition overheads.

SGX protects the confidentiality and integrity for user programs with isolated and shielded memory region. However, it is vulnerable to several side-channel attacks. Xu *et al.* [34] used the page fault channel to extract complete text documents and outlines of JPEG images from widely deployed application libraries on Haven and Ink Tag. Shinde *et al.* [15] showed that the page fault side-channel has sufficient channel capacity to extract bits of encryption keys from commodity implementations of cryptographic routines of OpenSSL and Libgcrypt shielded in SGX enclave and then propose PF-obliviousness and design deterministic multiplexing approach that eliminates information leakage via page fault channel to prevent this type side-channel. Déjá Vu [16] provides a software framework that enables a shielded execution to detect privileged side-channel attacks to SGX using by a trustworthy reference-clock based on transaction synchronization extensions, a hardware implementation of transactional memory.

In this paper, we focus the attestation mechanism based on SGX. Although side-channel attacks are important security threats for SGX, some current defense mechanisms [14]–[16] have been proposed to resolve this issue.

IX. Conclusion

In this paper, we have proposed a security-enhanced attestation approach for verifying the trust of RT. In our system, the attestation service including the measurement module and attestation module is divided into trusted and untrusted parts. The trusted parts are isolated in different enclaves and protected from malicious insider attacker. The sensitive data including keys and policy are also sealed in EPC. In addition, we have presented the policy-based measurement, so that the administrator can achieve custom measurement through self-defining policies. Our experimental results and analysis showed that our approach brings little overhead while enhancing security of the attestation procedure.

Acknowledgment

The authors would like to thank G. Jianjun from Intel and L. Xin from Nationalz for their generous help, some valuable comments, and opinions on this paper.

References

[1] J. Wang *et al.*, "Poster: An E2E trusted cloud infrastructure," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, Scottsdale, AZ, USA, 2014, pp. 1517–1519.

[2] B. Yang, D.-G. Feng, Y. Qin, and Y.-J. Zhang, "Secure access scheme of cloud services for trusted mobile terminals using TrustZone," *Ruan Jian Xue Bao/J. Softw.*, vol. 27, no. 6, pp. 1366–1383, 2016.

[3] J. Lyle and A. Martin, "On the feasibility of remote attestation for Web services," in *Proc. Int. Conf. Comput. Sci. Eng.*, Vancouver, BC, Canada, 2009, pp. 283–288.

[4] Y. M. Yussoff, H. Hashim, and M. D. Baba, "Identity-based trusted authentication in wireless sensor networks," *Int. J. Comput. Sci. Issues*, vol. 9, no. 3, p. 230, 2012.

[5] K. E. Defrawy, A. Francillon, D. Perito, and G. Tsudik, "Smart: Secure and minimal architecture for (establishing a dynamic) root of trust," in *Proc. Isoc*, San Diego, CA, USA, 2012.

[6] P. Koeberl, S. Schulz, A.-R. Sadeghi, and V. Varadharajan, "TrustLite: A security architecture for tiny embedded devices," in *Proc. Eur. Conf. Comput. Syst.*, Amsterdam, The Netherlands, 2014, Art. no. 10.

[7] *TPM Main Specification Level 2 Version 1.2, Revision 116*. Accessed: Sep. 15, 2017. [Online]. Available: https://trustedcomputinggroup.org/tpm-1-2-protection-profile/

[8] (2014). *Trusted Platform Module Library, Family 2.0, Revision 01.16*. [Online]. Available: http://www.trustedcomputinggroup.org

[9] "Security technology. Building a secure system using TrustZone technology," ARM Tech. White Paper, 2009.

[10] J. S. Jang, S. Kong, M. Kim, D. Kim, and B. B. Kang, "SeCReT: Secure channel between rich execution environment and trusted execution environment," in *Proc. NDSS*, 2015.

[11] L. Tan and J. Chen, "Remote attestation project of the running environment of the trusted terminal," *J. Software*, vol. 25, no. 6, pp. 1273–1290, 2014.

[12] A. Francillon, Q. Nguyen, K. B. Rasmussen, and G. Tsudik, "Systematic treatment of remote attestation," Cryptol. ePrint Archive, Tech. Rep. 713, 2012.

[13] *Open Attestation*. [Online]. Available: https://01.org/zh/openattestation?langredirect=1

[14] E. Brickell, G. Graunke, M. Neve, and J.-P. Seifert, "Software mitigations to hedge AES against cache-based software side channel vulnerabilities," Cryptol. ePrint Archive, Tech. Rep. 2006/052, 2006.

[15] S. Shinde, Z. L. Chua, V. Narayanan, and P. Saxena, "Preventing page faults from telling your secrets," in *Proc. ACM Asia Conf. Comput. Commun. Security*, Xi'an, China, 2016, pp. 317–328.

[16] S. Chen, X. Zhang, M. K. Reiter, and Y. Zhang, "Detecting privileged side-channel attacks in shielded execution with Déjá Vu," in *Proc. ACM Asia Conf. Comput. Commun. Security*, Abu Dhabi, UAE, 2017, pp. 7–18.

[17] R. Sinha, S. Rajamani, S. Seshia, and K. Vaswani, "Moat: Verifying confidentiality of enclave programs," in *Proc. ACM Sigsac Conf. Comput. Commun. Security*, Denver, CO, USA, 2015, pp. 1169–1184.

[18] F. McKeen *et al.*, "Innovative instructions and software model for isolated execution," in *Proc. HASP ISCA*, 2013, pp. 10–15.

[19] M. Henson and S. Taylor, "Memory encryption: A survey of existing techniques," *ACM Comput. Surveys*, vol. 46, no. 4, pp. 1–26, 2014.

[20] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia, "Privacy-preserving group data access via stateless oblivious ram simulation," in *Proc. 23rd Annu. ACM SIAM Symp. Discr. Algorithms (SODA)*, 2012, pp. 157–167.

[21] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions," in *Proc. HASP ISCA*, 2013, pp. 11–17.

[22] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd Int. Workshop Hardw. Archit. Support Security Privacy*, vol. 13. 2013.
[23] *Software Guard Extensions Evaluation SDK for Linux* OS User Guide*. Accessed: Sep. 15, 2017. [Online]. Available: http://www.intel.com
[24] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proc. 11th ACM Conf. Comput. Commun. Security*, Washington, DC, USA, 2004, pp. 132–145.
[25] J. Camenisch *et al.*, "One TPM to bind them all: Fixing TPM 2.0 for provably secure anonymous attestation," in *Proc. IEEE Symp. Security Privacy (SP)*, San Jose, CA, USA, 2017, pp. 901–920.
[26] "The trusted execution environment: Delivering enhanced security at a lower cost to the mobile market," Glob. Platform White Paper, pp. 1–26, 2011.
[27] "Get into the zone: Building secure systems with ARM TrustZone technology," White Paper, vol. 160, 2013.
[28] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, "Providing root of trust for ARM TrustZone using on-chip SRAM," in *Proc. 4th Int. Workshop Trustworthy Embedded Devices*, Scottsdale, AZ, USA, 2014, pp. 25–36.
[29] N. Santos, H. Raj, S. Saroiu, and A. Wolman, "Using ARM TrustZone to build a trusted language runtime for mobile applications," *ACM SIGARCH Comput. Archit. News*, vol. 42, no. 1, pp. 67–80, 2014.
[30] W. Li, H. Li, H. Chen, and Y. Xia, "Adattester: Secure online mobile advertisement attestation using TrustZone," in *Proc. 13th Annu. Int. Conf. Mobile Syst. Appl. Services*, Florence, Italy, 2015, pp. 75–88.
[31] F. Schuster *et al.*, "VC3: Trustworthy data analytics in the cloud using SGX," in *Proc. 36th IEEE Symp. Security Privacy*, San Jose, CA, USA, May 2015, pp. 38–54.
[32] A. Baumann, M. Peinado, and G. Hunt, "Shielding applications from an untrusted cloud with haven," in *Proc. 11th USENIX Symp. Oper. Syst. Design Implement.*, Broomfield, CO, USA, Oct. 2014, pp. 267–283.
[33] S. Arnautov *et al.*, "SCONE: Secure Linux containers with Intel SGX," in *Proc. 12th USENIX Conf. Oper. Syst. Design Implement.*, Savannah, GA, USA, 2016, pp. 689–703.
[34] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *Proc. Security Privacy*, San Jose, CA, USA, 2015, pp. 640–656.

Juan Wang (M'12) received the B.S. and M.S. degrees in computer science and the Ph.D. degree in information security from Wuhan University, Wuhan, China, in 1998, 2004, and 2008, respectively.

She is currently an Associate Professor with the Computer Science Department, Wuhan University. In 2010, she was a Visiting Scholar with Arizona State University, Tempe, AZ, USA. Her research has been supported by NSF and the National Basic Research Program of China (973 Program). She has authored and co-authored over 30 papers and holds 10 patents in security areas. Her current research interests include trusted computing, hardware-based security protection, and cloud security.

Dr. Wang was a recipient of Hubei Science and Technology Progress Award.

Zhi Hong is currently pursuing the master's degree with the Computer Science Department, Wuhan University, Wuhan, China.

His current research interests include trusted computing and Software Guard Extension.

Yuhan Zhang is currently pursuing the master's degree with the Computer Science Department, Wuhan University, Wuhan, China.

Her current research interest includes trusted computing. She is currently focusing on the use of Software Guard Extension in the area of trusted computing.

Yier Jin (M'13) received the B.S. and M.S. degrees in electrical engineering from Zhejiang University, Hangzhou, China, in 2005 and 2007, respectively, and the Ph.D. degree in electrical engineering from Yale University, New Haven, CT, USA, in 2012.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA. His current research interests include trusted embedded systems, trusted hardware intellectual property (IP) cores, and hardware-software co-protection on computer systems. He proposed various approaches in the area of hardware security, including the hardware Trojan detection methodology relying on local side-channel information, the post-deployment hardware trust assessment framework, and the proof-carrying hardware IP protection scheme. He is also interested in the security analysis on Internet of Things (IoT) and wearable devices with particular emphasis on information integrity and privacy protection in the IoT era.

Dr. Jin was a recipient of the DoE Early CAREER Award in 2016 and the Best Paper Award of DAC'15, ASP-DAC'16, and HOST'17.