Hindawi Publishing Corporation
Scientific Programming
Volume 2016, Article ID 4516596, 12 pages
http://dx.doi.org/10.1155/2016/4516596

Research Article

Research on Linux Trusted Boot Method Based on Reverse Integrity Verification

Chenlin Huang,[1] Chuanwang Hou,[1] Huadong Dai,[1] Yan Ding,[1] Songling Fu,[2] and Mengluo Ji[3]

[1]*School of Computer, National University of Defense Technology, Changsha, Hunan 410073, China*
[2]*College of Polytechnic, Hunan Normal University, Changsha, Hunan 410073, China*
[3]*Department of Computer and Information Engineering, Luoyang Institute of Science and Technology, Luoyang, Henan 471023, China*

Correspondence should be addressed to Huadong Dai; daihuadong@kylinos.cn

Received 25 February 2016; Accepted 8 May 2016

Academic Editor: Florin Pop

Trusted computing aims to build a trusted computing environment for information systems with the help of secure hardware TPM, which has been proved to be an effective way against network security threats. However, the TPM chips are not yet widely deployed in most computing devices so far, thus limiting the applied scope of trusted computing technology. To solve the problem of lacking trusted hardware in existing computing platform, an alternative security hardware USBKey is introduced in this paper to simulate the basic functions of TPM and a new reverse USBKey-based integrity verification model is proposed to implement the reverse integrity verification of the operating system boot process, which can achieve the effect of trusted boot of the operating system in end systems without TPMs. A Linux operating system booting method based on reverse integrity verification is designed and implemented in this paper, with which the integrity of data and executable files in the operating system are verified and protected during the trusted boot process phase by phase. It implements the trusted boot of operation system without TPM and supports remote attestation of the platform. Enhanced by our method, the flexibility of the trusted computing technology is greatly improved and it is possible for trusted computing to be applied in large-scale computing environment.

1. Introduction

With the boom of Internet, the lack of a trustworthy infrastructure has been a barrel for the healthy development of modern computing systems. More and more threats are introduced due to the design flaws in software and hardware, the improper authorization and authentication for legal users, the abusing use of resources, and so forth. The key to solve these problems is to build a trustworthy computing environment, where the safety of end system is well designed and can be verified and trusted. Trusted computing technology proposed by the Trusted Computing Group (TCG) is one of the main practical efforts to achieve this goal. Trusted computing architecture is based on the trusted hardware, Trust Platform Module (TPM), and realizes transitive trust through the constant trust metric in the progress of system boot process to build a trusted computing environment.

Trusted platform module TPM and the related software are introduced in trusted computing platform technology to be as trusted root of the system, through the trust transfer process to ensure the credibility of computing platforms and applications and to improve the security of the terminal platform. However, in order to support a variety of security features in TCG specifications, a special trusted hardware TPM is required to be deployed in the mainboard, which has become a main barrier limiting the popularization of the trusted computing platform technology. TPM is the base of the trust chain and the trusted root throughout the trusted boot process, which records and transfers trusted states in end system. However, the TPM chips are not yet widely deployed in most computing devices so far, thus limiting the applied scope of trusted computing technology. It is almost impossible to implement an overall trusted network computing environment due to the hardware barrel.

To fix the problem of lacking trusted hardware in existing computing platform, an alternative security hardware, USBKey, is introduced in this paper to simulate the basic functions of TPM and a new reverse USBKey-based integrity verification model is proposed to implement the reverse integrity verification of the operating system boot process, which can achieve the effect of trusted boot of the operating system in end systems without TPMs.

We have designed and implemented a Linux trusted boot method based on reverse integrity verification, with which the integrity of data and executable files in the operating system are verified and protected during the trusted boot process phase by phase. It implements the trusted boot of operation system without TPM and supports remote attestation of the platform. Enhanced by our method, the flexibility of the trusted computing technology is greatly improved and makes it possible to be applied in large-scale computing environment.

2. The Trusted Boot in Operating Systems

2.1. Related Works. The trusted hardware, TPM (Trusted Platform Module), plays a key role in trusted computing, working as the base of trusted computing architecture and the core to enhance the credibility of the general-purpose computing platforms and networks. At present, the core standard is TPM 2.0 [1]. TPM functions as a trusted root of trusted computing platform, providing key cryptographic functions and protected storage space which are necessary to build trusted computing environment by coordinating with other trusted computing software and hardware.

Given the limitations in TPM's architecture and cryptographic algorithms, new trusted computing architectures Trusted Cryptography Module (TCM) [2] and Trusted Platform Control Module (TPCM) [3] have been proposed by scholars in China as official standards. Double certificate structure is designed in TCM, and Chinese government approved cryptographic algorithms are supported besides commercial ones. TPCM is capable of performing active control and trusted measurement on hardware level, which controls the trusted computing base (TCB) in end systems. Compared with TPM, a more rigid and trustworthy architecture is implemented in TPCM.

Recently, trusted mobile computing platforms and trusted cloud services are research focuses in both industry and academia. Studies relating to trusted mobile computing platform focused on TrustZone in smart devices, trusted identity management, privacy protection, and other aspects. Ekberg et al. propose ObC (On-board Credentials) system which allows third party to develop and deploy credentials in the device based on TrustZone and provides a TEE (Trusted Execution Environment) function for application developers [4]; Sujeen and Periasami propose a data security and privacy protection technology based on TPMs [5]. Nyman et al. propose a new, rich authorization model to solve the traditional eID management problem by enhancing platform integrity verification and eID authentication based on TPM 2.0 [6]. Zhao et al. implement and evaluate trusted boot of the Trusted Execution Environment (TEE) based on ARM TrustZone with the on-chip SRAM Physical Unclonable Functions (PUFs) [7]. Santos et al. extend the concept of trusted computing to the background of Infrastructure as a Service (IaaS) and propose a trusted cloud computing platform (TCCP) for ensuring the confidentiality and integrity of computations [8]. But, users of cloud computing do not have currently appropriate tools for their verification of confidentiality, privacy policy, computing accuracy, and data integrity. Banirostam et al. propose Trusted Cloud Computing Infrastructure (TCCI) providing a closed execution environment for infrastructure service developers by a User Trusted Entity (UTE) [9]. Habib et al. propose a multifaceted trust management system architecture for cloud computing marketplaces and related approaches [10].

Since not all platforms are equipped with TPM module, the traditional TPM leads to the loss of mobility and flexibility due to the binding of user's identity with specific trusted platform. Accordingly, some scholars introduce some USBKey-based secure boot solutions based on the principles of separating platform and user certificate. To prevent the master boot record from easily being manipulated and infiltrated by bootkits, Müller et al. present Stark, which mutually authenticates the computer and the user in order to resist keylogging during boot and implements trust bootstrapping from a secure token (a USB flash drive) [11]. Based on UEFI (Unified Extensible Firmware Interface), Kushwaha proposes the creation of ESP (EFI System Partition) on the USBKey [12]. Since the ESP contains the boot loader and other critical codes for booting, the system always needs the USBKey to boot the system to a running state. Meanwhile, Microsoft's secure boot architecture Win 8 increases UEFI-based secure boot components and further enhances security on the traditional concept of trusted booting.

In summary, the more flexible and practical trusted boot technologies in operating system are being developed to be bound with new user application schemas and scenarios and to suit the development of mobile computing and cloud computing.

2.2. Trusted Boot in Linux. Trusted boot is one of the core functions of trusted computing platform. With the support of trusted hardware, a trusted running environment for services and applications is built with the verification of the integrity of the whole hardware and software during system boot process.

The following three key points must be guaranteed during the trusted boot process.

(1) The chain of trust must be established sequentially. Before the transference of control rights, the executable entity must be measured by trusted computing base. It can only be loaded and gain control rights after its integrity is verified, which fulfills the procedure of establishing chain of trust. (2) All the metrics and calls involved in the process of the establishment of the trust chain will eventually be completed by TPM. (3) During the establishment of chain of trust, all important secret data involved including keys, premeasurement data, and verification data must be stored and sealed inside TPM. TPM is responsible for ensuring the integrity and confidentiality of the secret data. Unlike

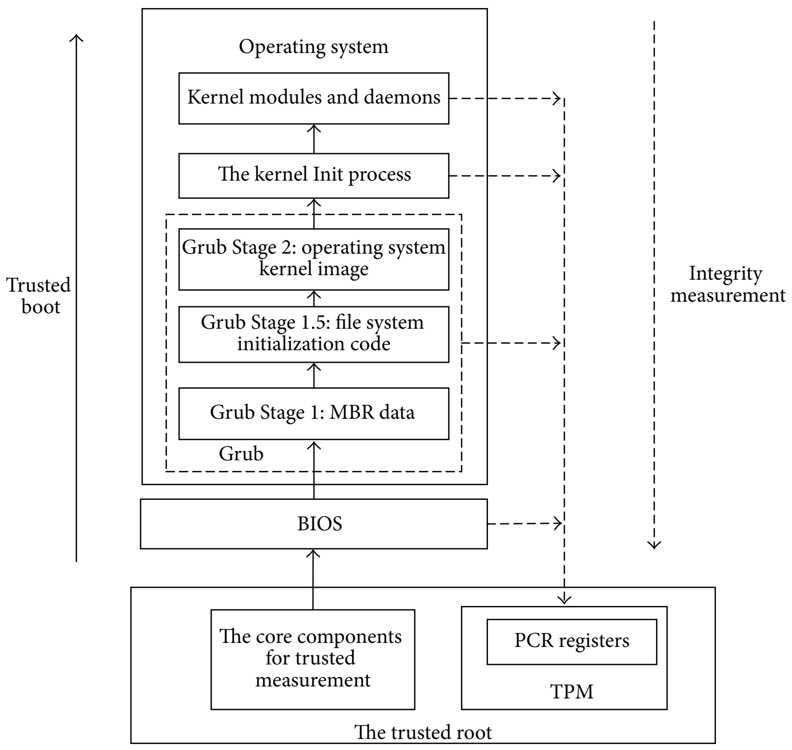
Operating system
Kernel modules and daemons
The kernel Init process
Grub Stage 2: operating system kernel image
Grub Stage 1.5: file system initialization code
Grub Stage 1: MBR data
Grub
BIOS
Trusted boot
Integrity measurement
The core components for trusted measurement
PCR registers
TPM
The trusted root

Figure 1: Linux trusted boot model.

removable storage devices or memory, there is no external call interface provided to access the secrete data in TPM, which ensures its confidentiality and credibility.

Taken the Linux trusted boot process based on TPM as an example, the trusted boot model in operating system is shown in Figure 1.

Trusted boot mainly includes two phases: the boot of hardware platform and the startup of operating system. The boot of hardware platform starts from the platform power on to the BIOS initialization and ends after the BIOS passes control rights to the boot loader. The reliability of hardware environment is measured and verified in this phase. The startup phase of the operating system begins with the loading of operating system loader from the main boot sector, and then the operating system kernel is loaded and ends till the running of the Init process. This stage is mainly responsible for checking the creditability of the system startup process and the operating system kernel. The trusted boot process of the startup phase of the operating system based on TPM is as follows.

Step 1. Trusted BIOS loads boot loader stored in Boot sector and then sends it to TPM to be measured and verified. Once TPM has verified its integrity, the boot program is loaded to memory 0000:7C00h, and then the BIOS passes control rights to the CPU to run the Boot program to further load operating system.

Step 2. TPM validates the operating system loader program, such as Grub in Linux. If the verification is successful, the Grub Stage 1 code in the master boot sector is loaded into memory and gains the trusted boot control to further load operating system kernel.

Step 3. The Grub Stage 1 continues trusted boot process by first validating Grub Stage 1.5 code with TPM, if it is successful, loads, and runs the code of the Stage 1.5 phase. At the end of this stage, the file system is mounted.

Step 4. The Grub Stage 2 code is verified by TPM and loaded by trusted Grub Stage 1.5. After successfully gaining control, it will verify the integrity of the configuration file "/boot/Grub/Grub.conf" in which the locations of the disk partitions, the kernel image, and virtual RAM disk file initrd are recorded.

Step 5. The Grub Stage 2 code opens the configuration file, reads the operating system kernel image, and tries to verify the integrity of the operating system kernel image by TPM. If it is successful, the operating system kernel image is loaded and gains control.

Step 6. Once the operating system kernel image is loaded, TPM will measure and verify the Init process. If the Init process is trusted, the kernel key data structures will be created and the kernel Init process will be loaded and take control.

Step 7. Firstly, the Init process determines the list of the kernel modules needed to be loaded and the daemons needed to

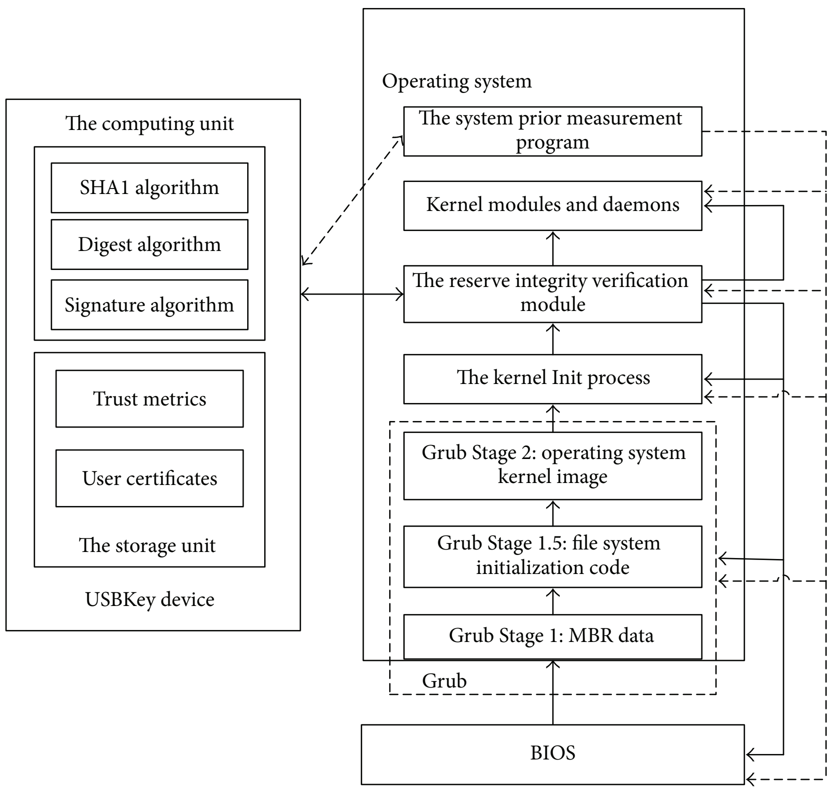

Figure 2: The reverse integrity verification model based on USBKey.

be created based on the system configuration. Then, it will measure and verify each kernel module and daemon with TPM module before they are loaded. Only the trusted kernel modules and daemons are run sequentially to guarantee that the initialized computing environment is trusted. At last, the Init process starts receiving users' inputs, and a trusted computer is ready to be used.

At this point, the trusted operating system boot process is over. As we can see, the traditional trusted boot process based on TPM is in forward direction, which means all measurements and verifications are strictly consistent with the operating system boot process. The chain of trust is established in a strict sequence.

3. A Linux Trusted Booting Method Based on Reverse Integrity Verification

3.1. A Reverse Integrity Verification Model with USBKey. As shown in Figure 2, in this paper, the operating system trusted boot method based on reverse integrity verification is implemented by the coordination of operating system kernel, the BIOS (firmware), and USBKey (USB smart card). The operating system kernel provides a basic software system running environment, including the drivers for system hardware and the construction of the system execution environment. In the process of the system boot, there are four parts in operating system kernel which are system boot Stage 1, Stage 1.5, Stage 2, and kernel modules. The BIOS firmware covers the initial stage of system boot, completes the initialization of hardware, and passes control to the boot loader. USBKey used in this method has built-in CPU, memory, Chip Operating System (COS), and internal safe data storage units, where secret data are stored, such as the user digital certificates and secret keys. There is also a sealed computing unit inside USBKey that supports cryptographic operations such as SHA1 algorithm, signature, authentication, data encryption and decryption, and data digest. All operations are calculated in the COS of USBKey which is totally safe to the outside world. As a trusted hardware, USBKey provides us a method to validate the integrity of data reliably.

Working as a trusted root, TPM provides support for the storage of trusted measurement and trusted state (PCR) in the establishment of chain of trust during trusted boot. It is also the starting point and foundation of system trusted boot and trust measurement. Especially in TPCM framework, the trusted hardware is regarded as trusted base and is the first functional hardware after powering on the system, so as to provide guarantee to keep the whole system in trusted states.

Similar to the TPM, USBKey, as widely used security hardware, also has built-in security guarantee, built-in trusted measurement algorithms, and built-in secure storage space. But in order to implement trusted measurement as TPM module does, USBKey still needs to get rid of the following two limitations: (1) Hardware driver needs to be loaded before USBKey could be used, which makes it unable to activate trusted measurement from powering on the system as required in TPCM. (2) There are no PCRs in the USBKey built-in storage. New software data structures have to be designed to simulate PCRs in TPM, so as to record system's trusted states and support trusted measurement during trusted boot.

To handle the first problem, we assume that a trusted prior measurement could be preceded by the kernel trusted measurement module while the system was in the initial trusted state. The trusted prior measurement generates foundation trust metrics for trusted measurement. To solve the second problem, a set of data structures are defined to simulate PCR registers in TPM which can be used to store the measurement values in the process of prior measurements and trusted boot. All measurement data are stored in safe storage area in USBKey.

The reverse integrity verification model based on USBKey is shown in Figure 2, in which the dotted line shows the process of the prior measurement and the solid line shows the process of the reverse integrity measurement in trusted boot process. There are five key elements in a trusted measurement model based on TPM: *PCR values for prior measurement, TPM, targets to be measured, PCR values, and verification results*. First of all, a *trust metric base* needs to be constructed as a credential reference library for later trusted boot in the prior measurement phase. During prior measurement, the entities in the system are measured by TPM following the sequence of system boot and the trust measurement values in PCRs are recorded into trust metric base. Then, in the process of system boot, TPM will measure the entities to be loaded, which are targets to be measured sequentially and compare the PCR values with the PCR values for prior measurement for consistency to reach the verification results.

In the reverse integrity verification model based on USBKey, the five key elements are *trust metrics for prior measurement, USBKey, targets to be measured, trust metrics, and verification results*. In our model, trust metrics refer to the trust measurement values generated by USBKey. Compared with the traditional trusted measurement model based on TPM, the key elements are similar, while the TCB and the trust measurement procedure differ a lot. The combination of "USBKey + trust metric base" functions as the root of trust instead of TPM only. The process of trusted boot is also divided into two phases: *the reverse integrity verification phase* and *the trusted boot phase*. After system starts up and successfully drives and loads USBKey, *the Reserve Integrity Verification Module* will initialize the reverse integrity verification phase. It will access the loaded system entities and measure and verify their integrity values with USBKey by comparing with the *trust metrics for prior measurement*. After phase one, a reliable system environment is verified and the trust foundation of the current system state is established. Then, *the trusted boot phase* will handle the rest of system boot the same as the process of trusted boot. *Together with USBKey*, *the Reserve Integrity Verification Module* functions as a TCB and makes sure the following loaded entities are trusted. Once an entity needs to be loaded, the module will call USBKey to measure the entity and verify its integrity by comparing with the *Trust Metrics for Prior Measurement*. Since only trusted entities are to be loaded and run, the whole system will be in a trusted state persistently.

Operating system trusted boot method based on reverse integrity verification contains the following steps.

(1) Prior Measurement. The initial system state can be assumed to be trusted. To start a prior measurement process, a regular system boot will be executed until starting to receive the input of user. Then, the System Prior Measurement Program will be loaded by the Init process and take control. The System Prior Measurement Program starts revising the whole boot process by measuring each stage during system boot sequentially. For every stage, the integrity of the relative entities is measured by USBKey; the measurement values are stored into USBKey safe storage unit to form *trust metric base*.

(2) Reverse Integrity Verification. For regular system boots after prior measurement, the reverse integrity verification can be performed by *the Reserve Integrity Verification Module* and USBKey. Once the operating system kernel is loaded and the USBKey is enabled, *the Reserve Integrity Verification Module* will be loaded into kernel and control the rest of trusted boot. The design of *the Reserve Integrity Verification Module* is similar to trusted software stack, and it is able to complete the integrity measurement of the target by cooperating with USBKey. First, it will read the prior measurement values for each boot stage from trust metric base, which are reference library for later trust verification. Then, each stage during system boot will be measured by USBKey. By referencing the trust metric base, the current integrity state of the system is verified by comparing the current measurement value with the relative prior measurement value. If all the loaded boot stages have passed reverse integrity verification, the current system is marked as in a trusted state. *The Reserve Integrity Verification Module* will continually be in charge to guarantee that all the modules, services, and applications loaded later are measured and verified by USBKey. Only trusted entities are to be loaded and run until the trusted boot succeeds and a trusted computing environment is established.

Following the system boot order, there are nine stages of data that are measured and verified in our method:

(1) The BIOS.
(2) The Grub Stage 1 data in the master boot sector.
(3) Grub Stage 1.5 data.
(4) Grub Stage 2 data.
(5) Grub configuration file.
(6) The kernel image file.
(7) The Init process data.
(8) The kernel modules to be loaded by the Init process based on the system configuration.
(9) The daemons to be loaded by the Init process based on the system configuration.

The nine stages can be divided into two parts: BIOS boot and operating system boot. The BIOS image needs to be measured and protected which includes codes for POST

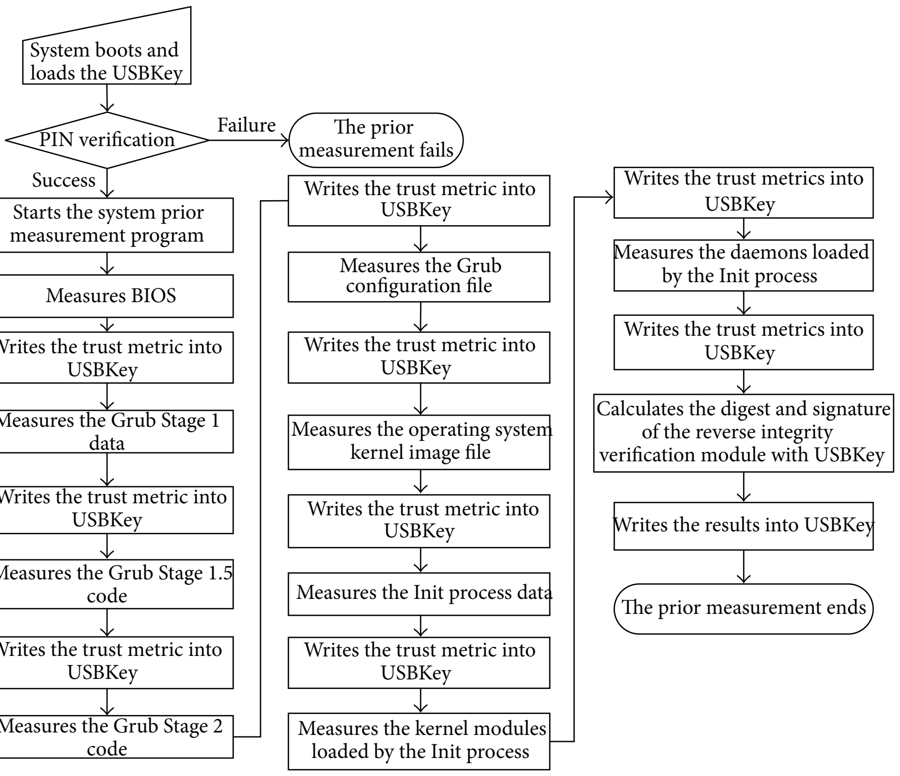
System boots and loads the USBKey
PIN verification
Failure
The prior measurement fails
Success
Starts the system prior measurement program
Measures BIOS
Writes the trust metric into USBKey
Measures the Grub Stage 1 data
Writes the trust metric into USBKey
Measures the Grub Stage 1.5 code
Writes the trust metric into USBKey
Measures the Grub Stage 2 code
Writes the trust metric into USBKey
Measures the Grub configuration file
Writes the trust metric into USBKey
Measures the operating system kernel image file
Writes the trust metric into USBKey
Measures the Init process data
Writes the trust metric into USBKey
Measures the kernel modules loaded by the Init process
Writes the trust metrics into USBKey
Measures the daemons loaded by the Init process
Writes the trust metrics into USBKey
Calculates the digest and signature of the reverse integrity verification module with USBKey
Writes the results into USBKey
The prior measurement ends

FIGURE 3: The design of the prior measurement for the reverse integrity verification.

(Power-On Self-Test), system hardware initialization, and loading operating system boot loader. The operating system boot phase contains Grub Stage 1, Grub Stage 1.5, Grub Stage 2, the kernel modules and daemons loaded by the Init process.

In the method, the System Prior Measurement Program and the Reverse Integrity Verification Module are located in the operating system kernel, whose legitimacy and integrity are protected by USBKey signature. They can be loaded if and only if USBKey PIN verification and signature authentication are success. In this paper, SHA1 algorithm in USBKey is applied to perform trust measuring.

3.2. The Design of Prior Measurement for Reverse Integrity Verification. The different stages of system boot are premeasured by USBKey. The trust metric base is constructed by collecting system prior measurement data, which are used to support trust measurement with USBKey in the process of the trusted boot. The design of the prior measurement for the reverse integrity verification is shown in Figure 3, the stages in the process of the operating system boot are measured by USBKey and the prior measurement values are stored in safe storage unit.

In prior measurement phase, a regular system boot will be executed until starting to accept the user input. Then, the System Prior Measurement Program will be loaded and begin measuring each stage of system boot sequentially. The trust measurement values are calculated by USBKey and are stored into USBKey safe storage unit. The trust metric base is constructed after prior measurement phase. As shown in Figure 3, the specific steps are as follows:

(1) Operating system boots up until receiving user input, that is, from powering on the system to the time the Init process is successfully running. Then, it inserts USBKey and verifies user's PIN. If successful, *the System Prior Measurement Program* is verified and loaded; otherwise the prior measurement fails.

(2) *The System Prior Measurement Program* reads the BIOS information and calls USBKey to execute trust measurement with SHA1 algorithm. The returned BIOS trust metric is stored in the safe storage unit in USBKey.

(3) *The System Prior Measurement Program* reads the Grub Stage 1 data from master boot sector and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(4) *The System Prior Measurement Program* reads the Grub Stage 1.5 code and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(5) *The System Prior Measurement Program* reads the Grub Stage 2 code and calls USBKey to execute trust measurement with SHA1 algorithm. The returned

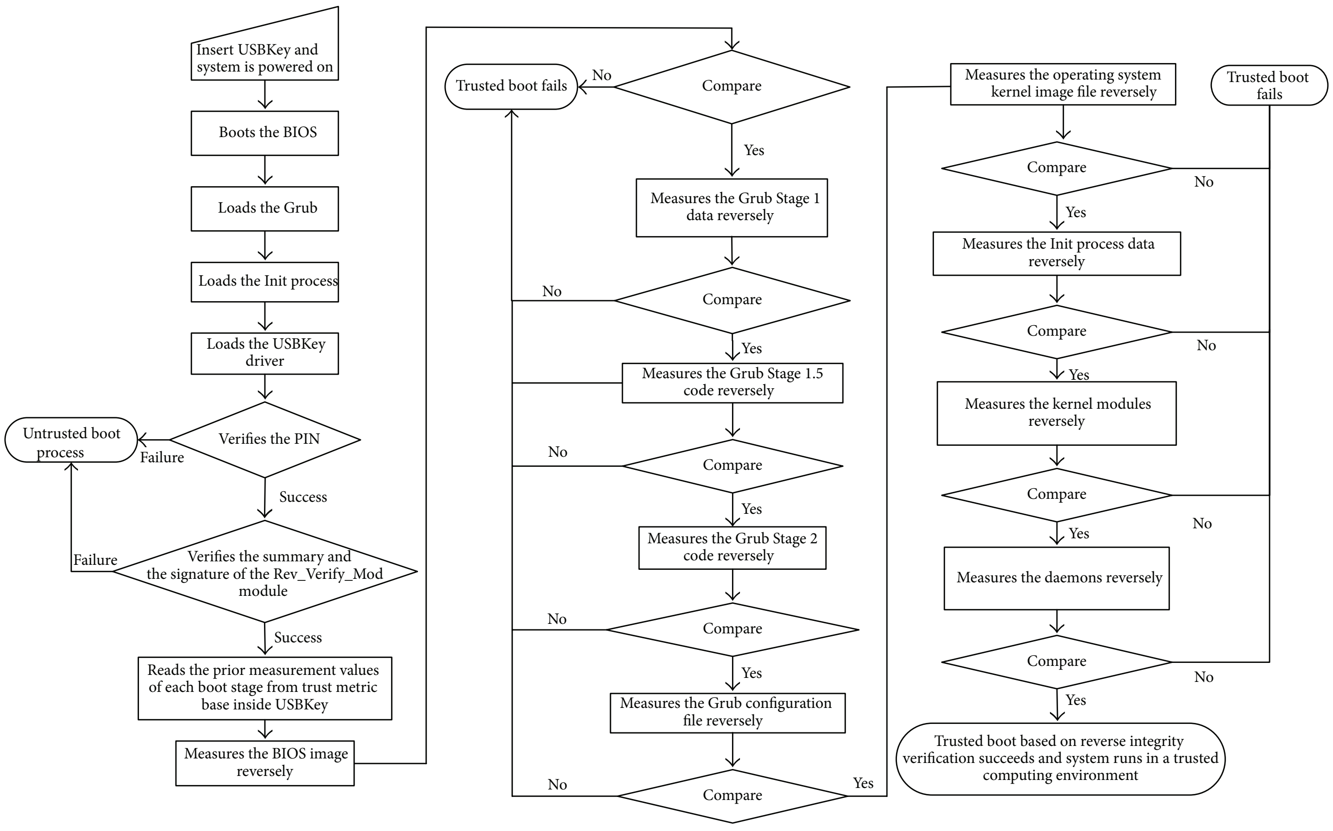
Insert USBKey and system is powered on
Boots the BIOS
Loads the Grub
Loads the Init process
Loads the USBKey driver
Verifies the PIN
Untrusted boot process
Failure
Success
Verifies the summary and the signature of the Rev_Verify_Mod module
Failure
Success
Reads the prior measurement values of each boot stage from trust metric base inside USBKey
Measures the BIOS image reversely
Compare
Trusted boot fails
No
Yes
Measures the Grub Stage 1 data reversely
Compare
Measures the Grub Stage 1.5 code reversely
Compare
Measures the Grub Stage 2 code reversely
Compare
Measures the Grub configuration file reversely
Compare
Measures the operating system kernel image file reversely
Compare
Measures the Init process data reversely
Compare
Measures the kernel modules reversely
Compare
Measures the daemons reversely
Compare
Trusted boot fails
Trusted boot based on reverse integrity verification succeeds and system runs in a trusted computing environment

Figure 4: The trusted boot of operating system with reverse integrity verification.

trust metric is stored in the safe storage unit in USBKey.

(6) *The System Prior Measurement Program* reads the Grub configuration file "/boot/Grub/Grub.conf," and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(7) *The System Prior Measurement Program* reads the operating system kernel image file and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(8) *The System Prior Measurement Program* reads the Init process data and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metric is stored in the safe storage unit in USBKey.

(9) *The System Prior Measurement Program* reads the kernel modules to be loaded by the Init process sequentially and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metrics are stored in the safe storage unit in USBKey.

(10) *The System Prior Measurement Program* reads the daemons to be loaded by the Init process sequentially and calls USBKey to execute trust measurement with SHA1 algorithm. The returned trust metrics are stored in the safe storage unit in USBKey.

(11) To protect TCB in the method, USBKey precedes the summary and signature verification on the Reverse Integrity Verification Module Rev_Verify_Mod with user digital certificate inside. The results as Rev_Verify_Mod's trust metrics are deposited in the safe storage unit in USBKey.

(12) The prior measurement process comes to an end.

3.3. The Trusted Boot of Operating System with Reverse Integrity Verification. After the completion of prior measurement, USBKey can reverse and verify the operating system booting process according to the stored prior metric values. To support USBKey-based verification, an operating system kernel module, *the Reverse Integrity Verification Module (Rev_Verify_Mod)*, is implemented to measure and verify the entities loaded and to be loaded modules and the modules by calling the USBKey. To demonstrate the detailed USBKey-based reverse integrity verification process, the USBKey-based Linux operating system trusted boot process is shown in Figure 4.

After the prior measurement, the trusted boot is enabled. The system is powered on and operating system kernel is loaded. Once the USBKey is on, the Rev_Verify_Mod will be verified and loaded. Then, the Rev_Verify_Mod reads the trust metric base from the safe storage unit in USBKey. At last, the Rev_Verify_Mod sequentially reads the boot information of each phase and carries out the trust measurement and verification by comparing the current trust metrics with the premeasured values. If they are not equal, the system state is set as distrusted and the corresponding trusted boot failure handler is activated. If they are equal, the current system state is set as trusted and trusted boot continues to the next phase. When all system boot phases are completed successfully, the operating system is verified to be trusted and the trusted boot succeeds.

As shown in Figure 4, the steps are as follows:

(1) Insert USBKey and system is powered on. The BIOS boots first, and then the operating system is loaded and boots until the Init process is successfully loaded and user interface is active.

(2) USBKey is enabled and the Init process requires user inputs PIN code to proceed to USBKey authentication.

(3) If USBKey authentication succeeds, it is ready to be used to proceed to trusted boot. First, the legitimacy and integrity of the Rev_Verify_Mod are verified by USBKey by comparing the calculated results with the prior measurement values. If they are identical, the authentication is successful. Then the Rev_Verify_Mod is loaded by operating system kernel, and the control is handled over to the Rev_Verify_Mod module.

(4) The Rev_Verify_Mod module reads the prior measurement values of each boot stage from trust metric base inside USBKey, which contains BIOS metric, Grub Stage 1 metric, Grub Stage 1.5 metric, Grub Stage 2 metric, the metric of the Grub configuration file, the operating system kernel image metric, Init process metric, kernel module metrics, and daemon metrics.

(5) The Rev_Verify_Mod module reads the BIOS information and measures its integrity with USBKey. By comparing the calculated results with the records of the BIOS metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(6) The Rev_Verify_Mod module reads the Grub Stage 1 data and measures its integrity with USBKey. By comparing the calculated results with the records of the Grub Stage 1 metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(7) The Rev_Verify_Mod module reads the Grub Stage 1.5 data and measures its integrity with USBKey. By comparing the calculated results with the records of the Grub Stage 1.5 metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(8) The Rev_Verify_Mod module reads the Grub Stage 2 data and measures its integrity with USBKey. By comparing the calculated results with the records of the Grub Stage 2 metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(9) The Rev_Verify_Mod module reads the Grub configuration file and measures its integrity with USBKey. By comparing the calculated result with the records of the Grub configuration file metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(10) The Rev_Verify_Mod module reads the operating system kernel image file and measures its integrity with USBKey. By comparing the calculated result with the records of the operating system kernel image file metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(11) The Rev_Verify_Mod module reads the Init process file and measures its integrity with USBKey. By comparing the calculated result with the records of the Init process file metric, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(12) The Rev_Verify_Mod module reads the data of the kernel modules to be loaded and measures its integrity with USBKey. By comparing the calculated results with the records of the kernel modules metrics, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(13) The Rev_Verify_Mod module reads the data of the daemons to be loaded and measures its integrity with USBKey. By comparing the calculated results with the records of the daemons metrics, the trustworthy state is judged. If they are equal, then trusted measurement succeeds and the current system is marked as in a trusted state; otherwise, the current system is marked as distrusted.

(14) Operating system boot is complete. During the process of the above reverse integrity verifications, the chain of trust is established by loading order. If any step encounters a verification failure, the system trusted boot fails and the system state is set to be distrusted and the corresponding trusted boot

failure handler will be called which will normally suspend the system boot process or switch to the distrusted boot process. If all integrity measurements are successful in all steps, then the operating system is set to trusted status, and the trusted boot succeeds.

(15) The operating system trusted boot based on reverse integrity verification succeeds and system runs in a trusted computing environment.

3.4. Comparative Analysis. USBKey is featured with low cost, safe, portable, convenient, and other characteristics. Compared with TPM, it can provide similar security functions with higher flexibility at the same time. They both have secure data storage space for sensitive data, such as trust metrics, digital certificates, and keys. Reading and writing operations on the safe storage space can only be achieved through the COS inside the hardware, which prevents the user from directly reading secret data. The user keys inside the safe hardware cannot be exported, which eliminates the possibility of the replication of user digital certificate or identity information. A variety of cryptographic algorithms are performed inside the safe hardware with the CPU inside. Operations such as encryption, decryption, and signature operations are performed within USBKey to ensure that the key will not appear in the computer memory, so as to prevent the fact that the user key may be intercepted by attackers. Therefore, by applying the USBKey and the reverse integrity verification method on operation system trusted boot, the system integrity can be guaranteed, and better flexibility and convenience are achieved.

In existing USBKey-based security enhancements methods, the USBKey is primarily used for user authentication, such as USBKey-based two-factor authentication, and data encryption and decryption. Our work in this paper extends the application field of USBKey. By constructing prior measurement libraries and reverse integrity verification modules, the USBKey in our method can simulate TPM and support key functions in trusted computing as trusted boot, trusted storage, and remote attestation.

The main advantages of the proposed method include the following:

(1) In the operating system trusted boot method based on the reverse integrity verification, the system is guaranteed by verifying the integrity of data and executable files in different booting stages. The verifications are not sensitive to loading sequence and are performed stage by stage to ensure that the system is in a trusted status. Compared with TPM-based trusted boot, USBKey based trusted boot is more compatible with application environment and is more flexibility and easier to be used in real computing systems.

(2) With the operating system trusted boot method based on the reverse integrity verification, the operating system trusted boot can be achieved on endpoints without TPMs, which greatly enhances the applicable scope of trusted computing technology. At the same time, the procedure for trusted boot is simplified, and trust measurements and verifications can be performed at any phase of system boot process through the reverse attestation method.

Compared with TPM-based trusted boot, the main disadvantage for our method is that the system is unprotected before the Rev_Verify_Mod module is loaded, which makes it possible to bypass the reversed verification procedure and boot a distrusted system. One solution is to enable USBKey and implement USBKey-based reversed verification in BIOS boot phase to achieve the lower level protection. Another option is to encrypt the file system with USBKey, which only can be decrypted after the system is verified to be trusted by the Rev_Verify_Mod module.

4. A Scenario for Remote Attestation Based on the Reverse Integrity Verification Method

4.1. Related Works about Remote Attestation. Remote attestation is one of the core functions in trusted computing. Users are able to authenticate the identity of the target platform and measure and verify the integrity of the trusted computing platform with remote attestation by using TPMs or TCMs.

Remote platform authentication is one of the key mechanisms in trusted computing, whose purpose is to prove the identity of a remote entity by exchanging and verifying a series of certificates with TPM. In 2004, Brickell and other scholars proposed TPM-based direct anonymous attestation (DAA) program [13], which utilized zero-knowledge proof and group signature technology to prove the identity of the platform, but it was too complex to be implemented. Following DAA program, many new methods have been proposed to improve the efficiency and usability of DAA [14–18].

TCG defines a binary remote attestation protocol to attest the integrity of the remote platform with trusted hardware TPMs/TCMs. But it is argued for the overexposure of the configuration information about the hardware and software in local trusted computing platform. To overcome the flaw in binary remote attestation, Chen and other scholars firstly proposed a property based attestation protocol, PBA protocol [19]. Later, PBA protocol was further studied, and more remote attestation methods based on properties were proposed [20–24]. So far, property based remote attestation solution has currently been viewed as being the most valuable and prosperous for remote attestation, which overcomes the problems in binary remote attestation such as complexity, privacy, misuse of proof, and other defects.

In China, the Trusted Connection Architecture (TCA) [25] is proposed as a standard remote attestation protocol which is an improvement of TCG's Platform Trusted Service (PTS). The basic authentication model in TCA is shown in Figure 5.

Before establishing a trusted network connection, Access Requester (AR) and Access Controller (AC) must separately load PTS by calling the specific platform binding functions. Then the bidirectional user authentication protocol is performed, in which Policy Management (PM) acts as a Trusted Third Party (TTP). Platform A and Platform B collect

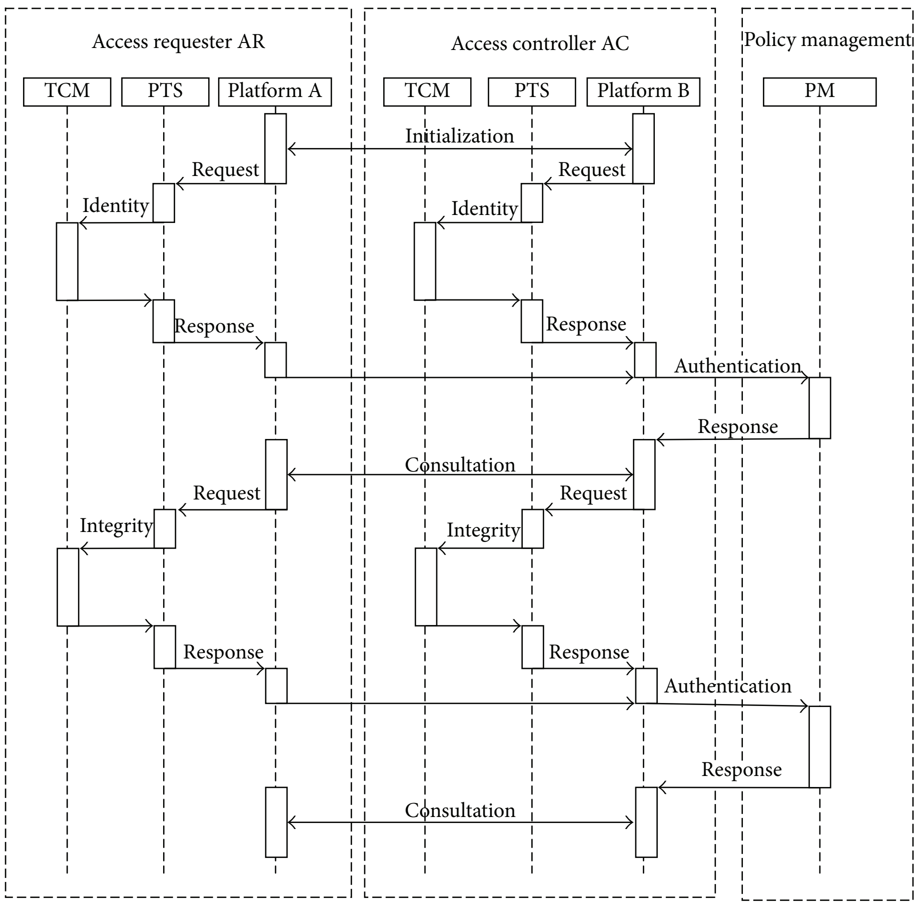

Figure 5: The basic authentication model in TCA.

integrity information with PTS protocol and then send the integrity information to PM. Finally the integrity of Platform A and Platform B is attested by PM.

4.2. A Scenario for Remote Attestation Based on the Reverse Integrity Verification Method in Hybrid Trusted Network. The remote attestation of the platforms relies on TPM chips; therefore, it is hard for PTS to be widely used in practical network environment since most of the servers and the terminal nodes are not equipped with TPM/TCM chips. The USBKey-based reverse integrity verification method in this paper can simulate TPM to measure trusted boot process and verify the trust state of the node. The system trust metrics and simulated PCR values are stored inside safe hardware with same data structures, which can be used to support remote attestation with other trusted nodes with TPM. Therefore, it is possible to deploy USBKey-based reverse integrity verification mechanism on nodes without TPMs and build a hybrid trusted network environment supporting remote attestation on all nodes.

In a hybrid trusted network environment, bidirectional remote attestation protocol between TPM-based trusted nodes and USBKey-based trusted nodes is designed as follows.

As shown in Figure 6, a new PTS protocol is designed with only a few simple modifications upon the original one to mask the differences between TPM/TCM and USBKey. A transparent layer is implemented to support both TPM/TCM and USBKey in remote attestation. Both PCR values generated by TPM/TCM and simulated PCR values generated by USBKey are collected as trust evidence. Thereby bidirectional remote attestation between TPM-based trusted nodes and USBKey-based trusted nodes can be achieved, which greatly reduces the cost of building an enterprise trusted network.

5. Conclusion

Aiming at the limitations in deploying and using TPMs in practical applications, we propose USBKey-based reverse integrity verification model which uses the widely used USBKey to establish chain of trust instead of TPM. The detailed design and implementation for this method are presented in Linux platform. The method supports the trusted boot of the Linux operating system. The PCR values are simulated to support the bidirectional remote attestation and trusted network connections between TPM-based trusted nodes and USBKey-based trusted nodes.

The proposed method greatly reduces the threshold for applying trusted computing technology. It has a wide

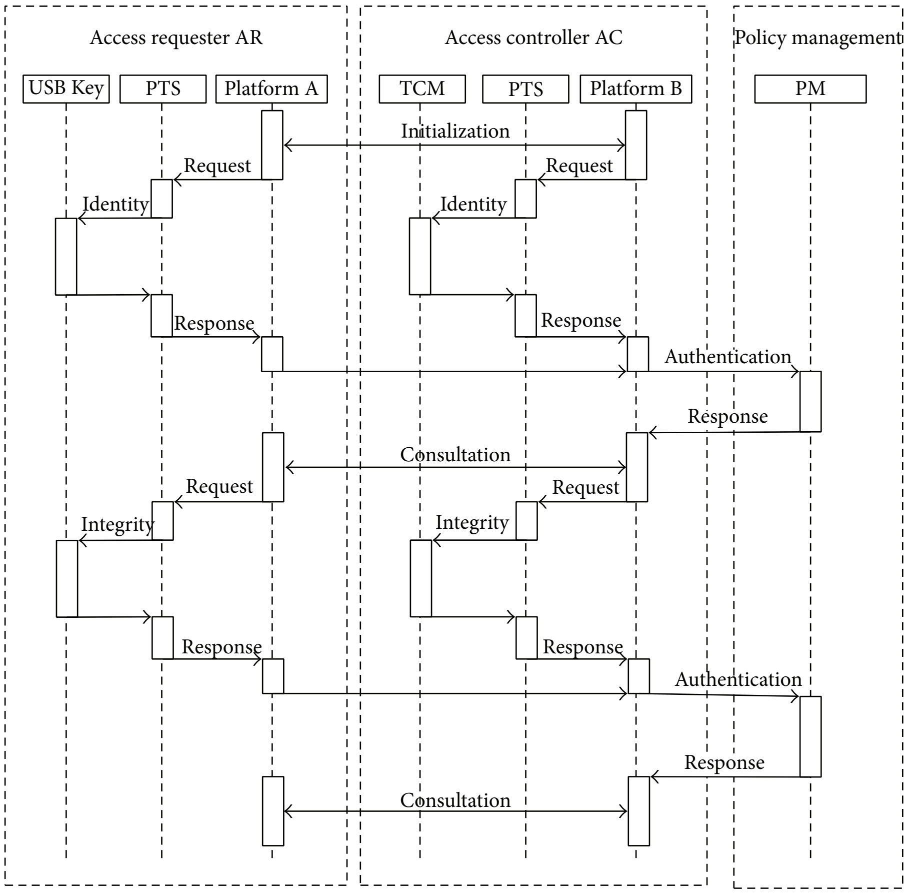

Figure 6: Remote attestation in hybrid trusted network.

prospective of applications and contributes a lot to the popularization of trusted computing technology in real enterprise environment.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant nos. 61303191 and 61502510), the HGJ Major Project of China under Grant no. 2015ZX01040301, and Foundation of Science and Technology on Information Assurance Laboratory (no. KJ-14-107).

References

[1] Trusted Computing Group, Trusted Platform Module Specification [EB/OL], http://www.trustedcomputinggroup.org/developers/trusted_platform_module/specifications.

[2] China TCM Unit, http://www.ztcia.com.

[3] X. Zhang and C. Shen, "A novel design of trusted platform control module," *Geomatics and Information Science of Wuhan University*, vol. 33, no. 10, pp. 1011–1014, 2008.

[4] J.-E. Ekberg, K. Kostiainen, and N. Asokan, "Trusted execution environments on mobile devices," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*, pp. 1497–1498, Berlin, Germany, November 2013.

[5] R. S. Sujeen and S. Periasami, "Verifying trusted code execution using ARM trustzone," *International Journal of Computer Science and Network Security*, vol. 13, no. 10, pp. 41–46, 2013.

[6] T. Nyman, J. E. Ekberg, and N. Asokan, "Citizen electronic identities using TPM 2.0," in *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices (TrustED '14)*, pp. 37–48, Scottsdale, Ariz, USA, November 2014.

[7] S. Zhao, Q. Zhang, G. Hu, Y. Qin, and D. Feng, "Providing root of trust for ARM TrustZone using on-chip SRAM," in *Proceedings of the 4th International Workshop on Trustworthy Embedded Devices*, pp. 25–36, ACM, Scottsdale, Ariz, USA, November 2014.

[8] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proceedings of the Conference on Hot Topics in Cloud Computing (HotCloud '09)*, p. 3, June 2009.

[9] H. Banirostam, A. Hedayati, A. K. Zadeh, and E. Shamsinezhad, "A trust based approach for increasing security in cloud computing infrastructure," in *Proceedings of the 15th International Conference on Computer Modelling and Simulation (UKSim '13)*, pp. 717–721, IEEE, Cambridge, UK, April 2013.

[10] S. M. Habib, S. Ries, M. Mühlhäuser, and P. Varikkattu, "Towards a trust management system for cloud computing marketplaces: using CAIQ as a trust information source,"

Security and Communication Networks, vol. 7, no. 11, pp. 2185–2200, 2014.

[11] T. Müller, H. Spath, R. Mäckl, and F. C. Freiling, "Stark," in *Financial Cryptography and Data Security*, pp. 295–312, Springer, Berlin, Germany, 2013.

[12] A. S. Kushwaha, "A trusted bootstrapping scheme using USB key based on UEFI," *International Journal of Computer and Communication Engineering*, vol. 2, no. 5, pp. 543–546, 2013.

[13] E. Brickell, J. Camenisch, and L. Chen, "Direct anonymous attestation," in *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS '04)*, pp. 132–145, ACM, 2004.

[14] D. Bernhard, G. Fuchsbauer, E. Ghadafi, N. P. Smart, and B. Warinschi, "Anonymous attestation with user-controlled linkability," *International Journal of Information Security*, vol. 12, no. 3, pp. 219–249, 2013.

[15] L. Chen and J. Li, "Flexible and scalable digital signatures in TPM 2.0," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS '13)*, pp. 37–48, November 2013.

[16] G. Proudler, L. Chen, and C. Dalton, "Direct Anonymous Attestation (DAA) in more depth," in *Trusted Computing Platforms*, pp. 339–352, Springer International, Berlin, Germany, 2014.

[17] L. Yang, J. Ma, W. Lou, and Q. Jiang, "A delegation based cross trusted domain direct anonymous attestation scheme," *Computer Networks*, vol. 81, pp. 245–257, 2015.

[18] B. Smyth, M. D. Ryan, and L. Chen, "Formal analysis of privacy in Direct Anonymous Attestation schemes," *Science of Computer Programming*, vol. 111, no. 2, pp. 300–317, 2015.

[19] L. Chen, R. Landfermann, H. Löhr, M. Rohe, A.-R. Sadeghi, and C. Stüble, "A protocol for property-based attestation," in *Proceedings of the 1st ACM Workshop on Scalable Trusted Computing (STC '06)*, pp. 7–16, ACM, November 2006.

[20] L. Chen, H. Löhr, M. Manulis et al., "Property-based attestation without a trusted third party," in *Information Security*, pp. 31–46, Springer, Berlin, Germany, 2008.

[21] J. Li, Y. Li, Y. Hu, H. Wang, and W. Liu, "An improved protocol for property-based attestation," in *Proceedings of the 32nd Chinese Control Conference (CCC '13)*, pp. 6343–6348, Xi'an, China, July 2013.

[22] V. Varadharajan and U. Tupakula, "Counteracting security attacks in virtual machines in the cloud using property based attestation," *Journal of Network and Computer Applications*, vol. 40, no. 1, pp. 31–45, 2014.

[23] Y. Liang, K. E. Guo, and J. Li, "The remote attestation design based on the identity and attribute certificates," in *Proceedings of the 11th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP '14)*, pp. 325–330, Chengdu, China, December 2014.

[24] X.-H. Yue and F. Zhou, "An efficient property-based attestation scheme with flexible revocation mechanisms," in *Proceedings of the IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW '12)*, pp. 1223–1230, Shanghai, China, May 2012.

[25] S. Changxiang and Z. Yuelei, "Trusted connect architecture," Chinese Standard GB/29828-2013, 2014.