TrustVP: Construction and Evolution of Trusted Chain on Virtualization Computing Platform

Xue Dongliang****, Wu Xiaolong***, Gao Yunwei**, Song Ying**, Tian Xinhui**, Li Zhaopeng*

**** College of Software Engineering of USTC, Hefei 230051, China, sea10366@mail.ustc.edu.cn
*** College of Computer Science, Sichuan University, Chengdu 610064, China, wxl@163.com
** State Key Laboratory of Computer Architecture, ICT, CAS, Beijing 100190, China,

{Gao Yunwei, Song Ying, Tian Xinhui}@ict.ac.cn
* USTC-Yale Joint Research Center for High-Confidence Software, Hefei 230026, China, zpli@ustc.edu.cn

Abstract—Trusted chain technology provides a good opportunity to guarantee software and data integrities on cloud computing platforms. However, trusted chain on current virtualization computing platforms expose some problems, such as non-continuous, difficult to evolve or customize, insecure to transmit for remote attestation. To address these issues, this paper proposes a new approach to construct, protect and update the trusted chain on virtualization computing platform. This approach constructs a complete trusted chain based on full-virtualization technology, ensures the security of the chain by using the seal and unseals features of Trusted Platform Module (TPM), and proposes an algorithm called TPRTM to update the chain. We also describe architecture and a prototype system implementation that can solve the problems mentioned above on current platform. Experimental results show that our method can guarantee the integrities of all customizable programs on the trusted chain while incurring only 2.23% performance degradation. Furthermore, the system has been deployed in a large-scale communication enterprise and the results reveal that our system is safe, stable, and easy to use.

Keywords: trusted chain, remote attestation, virtualization computing platform, trusted computing

I. INTRODUCTION

Security of the virtualization computing platform is gaining more and more attention and becomes the key issue for the popularity of cloud computing. As one effective solution, trusted computing technology is also being heavily researched. As pointed out in the specification of Trusted Computing Group (TCG): a trusted chain that starts from Core Root of Trust Measurement (CRTM) component and passes in turn through BIOS, OS Loader and OS Kernel can be constructed, all the programs in the trusted chain will be measured and verified. Meantime, when one program is maliciously altered, the boot process will fail due to this program’s destroyed integrity. Through integrity verification, a constructed trusted chain ensures the integrities of all the system resources and enhances the overall security of the system. However, in the virtualization environment, there are lots of problems about trusted chain. For example, the key steps of the boot process are not measured or verified. Lots of vulnerabilities exist during the transmission process of trusted chain. Updating and evolution of trusted chain are really difficult.

In previous work [4][5], GRUB STAGE1 which is the first part of Grub boot loader is not measured, thus trusted chain is in fact not complete. Meantime, the trusted chain is only limited to Linux environment. To mitigate the problem, [6] uses LILO as the boot loader which is not popular as GRUB, at the same time, trusted chain of the virtualization computing platform are also not concerned about. To measure programs, Trusted GRUB [7] and IMA [8] need read data from the disk twice. Thus an attacker has a chance to send malicious codes and these codes can result in security breach of transmission of trusted chain after entering the trusted chain.

Based on previous works, remote attestation is added into integrity verification process [9][10][11][12], and the legal measurement value of programs are stored in a remote server in advance. In the process of remote attestation all loading programs and their measurements are compared with the baseline-measurement-values in the server, the comparing consequence determines whether the platform is trusted. The direct result of this method is that we must change the baseline-measurement-value of programs in remote server. But, lots of complicated steps, such as authentication, key unregistration, key registration, issue of trust certificates, verification of signature, must be taken. So remote attestation brings the problem of updating or evolution difficulties of trusted chain, affects the flexible and secure maintenance of the chains.

In this paper, we provide a complete solution, TrustVP, for trusted chain on the virtualization platform. TrustVP ensures the secure transmission of trusted chain using the Non-volatile Storage of TPM to store the baseline-measurement-value of programs and using the function of Seal and Unseal of TPM to protect the measurement value of trusted chain. In addition, we implement a Trusted Program Record Table Manager (TPRTM) to sustain the chain more flexible, safer. The operation results of our prototype in a large communication company show that the process of construction and evolution of trusted chain has only 2.23% degradation, but ensures the integrity of key programs of this enterprise.
The main contributions of the paper are listed as follows:

1) We construct a complete trusted chain which involved physical machine bootstrap, Virtual Machine Monitor bootstrap, privileged domain (domain0) bootstrap and unprivileged domain (domainU) bootstrap.

2) We propose a measure-verify model that fits to virtualization computing platform in modern clouds. The innovation of this model is that it adds real-time verification procedure to the boot process based on the original measurement method, with this model our trusted chain can be maintained more securely.

3) We design a TPRTM to make programs on the trusted chain more customizable.

4) We design a new bootstrap approach that is compatible with attestation boot-mode and normal boot-mode for virtualization computing platform. In other words, this model can utmost customize the trusted chain.

The rest of the paper is organized as follows. The design of TrustVP, including background is described in section II. Section III discusses the implementation of TrustVP. The experimental and analytical results are presented in section IV. Finally, we summarize our work and discuss the future plan in section V.

II. BACKGROUND AND RELATED WORK

A trusted chain means that a program can be trusted only if its signature is valid, and if the signature is invalid, the previous software would not have executed it, and the trustworthiness of each layer is guaranteed by the hardware.

IMA [8] is a secure integrity measurement system for Linux. All executable contents that are loaded onto the Linux system are measured before execution and these measurements are protected by Trusted Platform Module. It does not concern about the virtualization platform. Aimin Yu[22] designed and implemented TCG-based remote attestation under the assumption that the TPM and hypervisor are secure and the privileged domain0 may be malicious, which is similar to us. However, theirs' trusted chain is independent of the domain0, while our trusted chain is a integrated chain from domain0 to domainU. B-IRP [25] proposes Batch Integrity Report Protocol (B-IRP) to overcome the performance bottleneck for Remote Attestation which is complicated and unnecessary for our native attestation. TARP [26] is a analogies protocol apply to Remote attestation based on TPM, and the difference is that it is used in wireless sensor networks. Furthermore, these works are based on remote attestation so that their verification is not as real-time as ours.

Terra [16] is a trusted computing architecture. In Terra architecture, a trusted virtual machine monitor can identify the contents of closed-box VMs to remote parties and allow them to trust it. Terra is also based on remote attestation so that verification is not as real-time as ours.

Copilot [17] provides a dynamic memory measurement mechanism by installing a memory monitoring co-processor on the motherboard. In Copilot, the hardware can hash the memory area that contains the kernel text and other key components through direct memory access (DMA). Paul England and Talha Tariq [21] explore a new model for trusted computing in which an existing fixed-function TPM is coupled with user application code running on a programmable smart card. These two works needs extra hardware equipments, however ours can be directly applied to off-shelf server.

Paul England describes three practical techniques for authenticating the code and other execution state of an operating system using the services of the TPM and a hypervisor in [18]. And in [19] he proposes a technique that allows a hypervisor to safely share a TPM among its guest operating systems. The design allows guests full use of the TPM in legacy-compliant or functionally equivalent form. These two works have certain reference value for our work, but our work mainly focuses on the integrities of full-virtualized VMs.

TPM-SIM [20] presents a simulation toolset for estimating the impact of Trusted Platform Modules (TPMs) on the performance of application that use TPM services in multi-core environment. Its optimization suggestions are beneficial to our performance tuning.

Since most application of TPM is base on the assumption that TPM is trusted, the TPM itself security is a concern. [23] Introduced power analysis attacks against these devices and architectures, and explore the feasibility of power analysis on the TPM's cryptographic system. And catching the Cuckoo [24] is also this case, it base on a protocol that uses an ordinary smart card to verify the platform boot integrity through TPM quote requests, and to verify TPM proximity by measuring TPM tick-stamp times required to answer the quotes. These works mainly focus on the weakness of the TPM hardware which is out of our concern, but cloud computing provides enough physical security so it is not a major concern of cloud computing. CERTICLOUD [27] is a novel approach for the protection of IaaS platforms that relies on the TPM to offer a secured and reassuring environment by two protocols: TCRR and VerifyMyVM. This work focus on the authentication and traceability of user deployed environment, but our work concerns about the integrities of the cloud computing platform, it is a complementary work for us.

III. SYSTEM DESIGN

Trusted chain has the following problems: non-continuous, difficult to evolve or update and insecure to transmit for remote attestation. TrustVP aims to resolve the above issues.

In subsection A, our group presents the overall architecture of the system and its details. The key part of the architecture is TPRTM which gives a method flexibly updating the trusted chain. In subsection B, we give the algorithm of TPRTM. In subsection C, we construct a complete trusted chain that just solves the problem of non-continuous chains. In subsection D, we propose an algorithm named Tool of Computing Integrity Verification Value (TCIVV) to compute the baseline-measurement-values of programs, which uses the idea of finite state machine.
These four parts connect tightly, so they as a whole resolve the existing problems of trusted chain on virtualization platform in modern clusters.

A. System Architecture Overview

The system architecture is shown in figure 1. The heavy gray parts are newly added modules based on the original Xen Full-Virtualization architecture, and the slight gray parts are some modules that we add to the original BIOS, GRUB, Xen and OS kernel. Firstly, we describe the main function of components in the architecture. Secondly, we discuss the working procedure of these components based on the architecture. Finally, we give a logic Implementation of architecture.

Figure 1  the prototype architecture

- **Function of Domain0 Components**

In domain0, the trusted chain is originated from BIOS block, and then to GRUB_STAGE1_5, and eventually transferred to the whole domain0 of virtualization computing platform.

a) GRUB

The main role of this component is to verify the important parts including xen, vmlinuz and initrd of domain0 at real time. We add our attestation codes in GRUB_STAGE1_5, GRUB_STAGE2, and hence we can verify the integrity of GRUB_STAGE1_5 running time as well as GRUB_STAGE2 loading time, then we continue to verify the integrity of xen kernel, vmlinuz and initrd during these programs’ loading time.

b) OS Kernel

Dynamic Link Library, Kernel Modules and Executable Programs are always injected into by malicious codes, and thus the integrity of these programs is performed by this component. We add our attestation codes at these programs’ loading time, if and only if the real-time measurement value of these programs consistent with baseline-measurement-value can the platform continue running.

c) vTPM Manager

The vTPM Manager of domain0 is responsible for creating and maintaining the vTPM instance of domain0, the instances from 1 to n in domain0 are one-to-one corresponding with virtual TPM1 to virtual TPMn in domainU. The vTPM Manager provides communication channels among vTPM instances, there is also communication channel between vTPM and hardware TPM.

d) TPRTM and TCIVV

TPRTM and TCIVV are vital parts in our architecture, we give the details in subsection B and subsection D respectively.

- **Function of DomainU Components**

In domainU, the trusted chain is originated from vTPM Manager in domain0, and then to GRUB_STAGE1, GRUB_STAGE1_5, GRUB.Stage1_5, and eventually transferred to all domainUs of virtualization computing platform. It should be noted that we give the function and implementation of Virtual BIOS and GRUB solely, the remainder of domainU component is consistent with domain0 corresponding component.

a) Virtual BIOS and GRUB

These two components closely connect to each other. In order to finish the integrity of GRUB_STAGE1, GRUB_STAGE1_5, we add BIOS interruption call at these programs, the function of these calls is to verify the integrity of these programs, but the response of call is implemented at Virtual BIOS. The creativity of this mode separates integrity function into two components under size limiting codes, because GRUB_STAGE1, GRUB_STAGE1_5 is only 512 bytes after compiling. In addition, the real-time characteristic of verifying the integrity of these programs is similar to domain0.

b) OS Kernel

c) TPRTM and TCIVV

- **Working Procedure of Architecture**

This architecture uses Full-Virtualization technology, and each domainU can take full advantage of functions and services of TPM, without modifying the kernel of domainU. The vTPM provides the same functions and services in domainU as hardware TPM in physical machine, but each domainU do not actually has hardware TPM.

As shown in figure1, the virtual TPM, which is outlined with dotted line, is simulation equipment created by a daemon process called QEMU in domain0. When a domainU user sends a TPM request, the Trap will be triggered, and then the Xen hypervisor will capture the Trap and send it to the QEMU process running in domain0. Since the vTPM Manager directly accesses hardware via TPM Driver, with this mechanism, all requests of TPM from each domainU are under protection of the hardware TPM.

TCIVV, which is run by platform managers in domain0 and domainU, produces configuration file named "Initial-Config.xml", whose form is shown in Figure 2 and its main function is to store the baseline-measurement-value of the programs. For domain0, the measure configuration file is the input of TPRTM Client. After getting the configuration file, TPRTM Client parses it, then executes the sealing operation of TPM, in fact, by the sealing operation, the TPRT can safely store into Non-volatile Storage area of TPM, During
this operation, data stream begins from TPRTM Client, goes through TPM Driver, and ends at hardware TPM. The sealed results are passed to TPRTM Server through socket, the server generates and saves the final TPRT. For domainU, the baseline-measurement-value configuration file is also the input of TPRTM Client, and it also has operations like that of domain0, but the final TPRT is generated and saved by the TPRTM Client of domainU, each domainU’s TPRTM Client communicates with domain0’s TPRTM Server by socket.

![Figure 2 Architecture of prototype](image)

- Logic Implementation of Architecture

Based on the Architecture of prototype, Figure 3 describes the Attestation Procedure of Trusted Chain on Platform through sequence diagram. (Note: LMP (L means dynamic library; M represents Module; P means the executable Program)). In the diagram, Call TPM means a BIOS interruption which applies to interface of TPM_UsealProgramList and Measure; however, the operation of Load, TPM_UsealProgramList, AttestProgram and Measure only signify logic implementation. The operation of Load indicates loading the programs in processing of bootstrap. The operation of TPM_UsealProgramList demonstrates sealing the TPRT which is stored in Non-volatile Storage of TPM, it is same as tpm_unseal described in subsection B, The operation of AttestProgram implies whether the measurement-value consistent with baseline-measurement-value. Above all, at OSLoader stage, the system Load programs of Xen, Vmlinux and Initrd, then Measure the hash value of these programs, ultimately, AttestProgram whether can continue to run, At stage of Linux Kernel bootstrap on virtualization platform, The Processing Principle is similar to OSLoader stage. The only differences is that we need the right place of Linux Kernel to add these operations.

![Figure 3 Attestation Procedure of Trusted chain on Platform](image)

B. TPRTM Algorithm

TPRTM plays the key role for the proposed architecture. We firstly give definition about tpm_seal and tmp_unseal functions, and then give the detailed description about TPRTM Algorithm.

**Definition 1**: Function tpm_seal is the seal function and tpm_unseal is the unseal function. Data structure buffer_in is the date set to be sealed or unsealed, and buffer_out is the date set generated by the seal function or unseal function. Permap is the data set of the current values of PCR. Dataauth and keyauth are the authorization string and authorization password for the TPM operations on a block of data. Tpm.dat is store the baseline-measurement-value of the programs on virtualization platform.

The buffer_in is transformed to buffer_out with the help of input arguments including permap, dataauth and keyauth. The tpm_unseal function can execute successfully if and only if permap is the same as the sealed value. The implementation algorithm of TPRTM is shown in Figure 4. Line 3, line 4, line 5 and line 6 show that if TPRTM does not execute for the first time, TPRTM client connects to the TPRTM server via the socket interface. Then the TPRTM server will copy the file tpm.data from a specific directory to the buffer and send it to the TPRTM client through shared memory. Line 8 and line 9 show that the buffer will be segmented to appropriate size and unsealed by the tpm_unseal function. The configuration file InitialConfig.xml is parsed by the REPLACE, DELETE and MERGE methods from line 11 to line 25 so that the list of trusted programs can be maintained by the administrator according to the requirements. And the replace, delete and merge methods may update the path, hash code and length of corresponding programs. Line 26 to line 29 split the blockbuf generated by parsing InitialConfig.xml and seal it to the sealedblockbuf, while the file tpm.dat is generated and sent to the TPRTM server. Note: if TPRTM run for the first time,
the program starts from line 11 and the file tpm.dat is generated in line 28. Or else the file tpm.dat that already exists is read in line 5. After that, each time TPRTM executes, the previous state of the system was sealed, and then the replace, delete and add operations are allowed. The characteristics of tpm_seal and tpm_unseal functions ensure the security of these updates.

Algorithm 2: TPRTM
Input: Initial Config.xml
Procedure
1. TPRTM Server Startup;
2. TPRTM Client Startup;
3. if TPRTM firstrun == FALSE
4. connect to TPRTM Server by Socket;
5. TPRTM Server read tpm.dat from non-volatile storage;
6. convert tpm.dat to a buffer;
7. send buffer to TPRTM Client by shared memory;
8. split buffer into appropriate size for tpm_unseal;
9. tpm_unseal(permmap, dataauth, keyauth, buffer, blobbuf);
10. endif
11. begin parse InitialConfig.xml;
12. if updateType == REPLACE
13. get current ID to blobbufApps[i]path;
14. get current Hashcode to blobbufHashcode;
15. refresh blobbufLength;
16. else if updateType == DELETE
17. memset(blobbufApps[i]path, 0);
18. memset(blobbufHashcode);
19. refresh blobbufLength;
20. else if updateType == MERGE
21. get new merged ID to blobbufApps[i]path;
22. get new merged Hashcode to blobbufHashcode;
23. refresh blobbufLength;
24. endif
25. end parse InitialConfig.xml
26. split blobbuf into appropriate size for tpe_seal;
27. tpm_seal(permmap, dataauth, keyauth, blobbuf, sealedBlobbuf);
28. write sealedBlobbuf to tpm.dat;
29. send tpm.dat to non-volatile storage of TPRTM Server.

Figure 4 The algorithm of TPRTM

C. Construct Trusted Chain on Virtualized Platform

Constructing a trusted chain is very important in trusted computing system. The CRTM is precondition to construct trusted chain. TrustVP treats the BIOS boot block as the CRTM, and based on CRTM TrustVP expands, and eventually extends to the whole virtualized platform. Specific trusted chain is shown in Figure 5.

Figure 5. Trusted chain of virtualization platform

The shadow parts in Figure 5 avoid trusted links missing, TrustVP adds the links of measuring and verifying trusted chain on the virtualization platform. Our trusted chain has two characteristics as follows.

On the one hand, we add trusted links of Xen kernel [13] and vTPM Manager which are not measured and verified in the previous work. In this paper, based on the Xen platform, we insure the trust of the VMM under the protection of hardware TPM, it fundamentally changes the hypothesis that VMM is trust. On virtualized platform, measuring and verifying VTPM Manager ensures that the trust links of domainU can more trusted.

On the other hand, TrustVP makes adequately use of full-virtualization technical to insure the domainU trusted chain links more trust when it is transmitted. Further more, it is reducing the measure hole of the trusted chain. Xen itself supports Para-virtualization, In order to support full-virtualization, Intel VT [15] technology enable xen completely support technical. Our system makes fully use of the advantages of full-virtualization. For the purpose of verifying the domainU GRUB_STAGE1, GRUB_START, GRUB_STAGE1_5, we use BIOS interruption call to add measure and verify function, it effectively resolves the problem of adding complex function in limited storage space.

D. TCIVV Algorithm

The remote attestation needs support of a trusted third party, however we store the integrity value of the program in local virtualization platform. It is also convenient to match baseline-measurement-value with the measurement-value when system real boot time. In this way it doesn’t need participation of the trusted third party, and reduce the possibility of exposing the various certificate and key. This algorithm makes full use of protection function of the TPM hardware, so it improves the security. Hence, how to get the baseline-measurement-value of integrity becomes a problem must to be solved, if and only if getting the baseline-measurement-value can enable us to process verification. Before the TCIVV algorithm is described, we firstly give the following definition.

Definition 2: $M = \{Q, I, move, S, F\}$ is a 5-tuple. $Q$ is a set of process state, which implies that the completed calculation values of the programs that have been measured form a state set. The expression, $\forall q_j \in Q, j \in [1, n]$ , implies that $q_j$ is got after computing the measurement value of program $j$; $I$ represents an input set, which implies the set of the program’s disk storage location. The expression, $\forall i_j \in I, i \in [1, n]$ , implies that the disk location of program $j$ is $i_j$; $S$ is a collection of initial state, which implies the initial state set prepared to do measure computation. The expression, $\forall s_j \in S, i \in [1, n]$ , signatures that the state of program $j$ is $s_j$ before measurement. move is a state conversion function, which is equate-to a mapping operation $S \times I \rightarrow Q$. For example, $move(s_j, i_j) = q_j$ means that the program $j$ through move can switch to next state $q_j$. 
when \( j_i \) is the input. \( q_j \) is the succeeding state of \( s_j \); \( F \) is a collection of termination state, which implies that the program has already finished computing the measurement value. The expression, \( \forall f_i \in F, \ i \in [1, n] \), signatures that the final state of program \( j \) is state \( f_j \); Figure 6 gives the algorithm.

Algorithm2:TCIVV

Input: \( Q = \emptyset, F = \emptyset, S = \{ s_0, s_2, ..., s_l, s_n \} \)
Output: \( F = \{ f_0, f_2, ..., f_l, f_n \} \)
Procedure:
1: for \( j = 1 \) to \( n \)
2: \( s_j \) addressing \( \rightarrow \) get start address installed in disk of \( s_j \);
3: \( s_j \) num \( \rightarrow \) get number of sectors \( s_j \) install needed;
4: \( q_j \) \( \rightarrow \) sha1(s_address, sj, snum);
5: \( Q \leftarrow Q \cup q_j ; \)
6: \( S \leftarrow S \leftarrow s_j \);
7: \( s_j \) addressing \( \rightarrow \) get start address installeed in disk of \( s_j \);
8: \( j \leftarrow j + 1 ; \)
9: endfor;
10: for \( j = 1 \) to \( n \)
11: get \( q_j \) from \( Q \);
12: \( f_j \leftarrow q_j \);
13: \( f_j \) value \( \leftarrow \) stored hash value of \( f_j \);
14: \( F \leftarrow F \cup f_j \);
15: \( j \leftarrow j + 1 ; \)
16: endfor.

Figure 6. The algorithm of TCIVV

As Shown in Figure 6, the operations of line 3 and line 4 get the start address of the program prepared to be measured, and the size of installation sector. These two values are used as the input argument of sha1 computing on line 5. The feature of sha1 is as followed: the recovery information can’t be get from message digest; two different messages (the program needed to be measured) can’t produce same digest; the message digest can be used to verify the integrity of data. Operations from line 1 to line 9 do all the sha1 computing of the program needed to be measured, and generate the middle state set Q. Operations from line 10 and line 16 get the hash values of every measured program, and store them in the final state set F in the end.

IV. EVALUATION

In this section, we evaluate TrustVP in two aspects: functional correctness and performance overhead. Subsection A gives results of function tests. Subsection B presents distribution and reason of performance overhead of TrustVP. Our experiment machine has an Infineon v1.2 TPM chip, dual-core Intel Xeon X6550 CPU at 2.33GHz, and 16GB of RAM. It runs Xen-4.1.0 virtual machine monitor, and uses SUSE Linux Enterprise Server 11 SP1 for x86 64-bit platforms as OS of privileged domain and guest domains, its boot loader is GRUB-0.97.

A. Functional Evaluation

Through experiments, we demonstrate that our system can build a complete continuous trusted chain. Assuming xen is malicious altered, as shown in experiments, this compromise can be detected promptly. In addition, our experiments show that some minor configuration and a few bad commands are enough to securely update the baseline metrics of trusted chains. Since no network transport is involved, so there is no trouble of network vulnerability.

> Computing Baseline-measurement-value

The following table shows the results of measuring xen-4.1.gx, vmlinuz-2.6.32.12, initrd-2.6.32.12, libc.so and other important programs by the TCIVV algorithm, which will serve as baseline-measurement-value for trust verification of boot process of the system. If any program is tampered before or during the boot processing, the value metrics must be different and the startup process will fail.

<table>
<thead>
<tr>
<th>Programs</th>
<th>Baseline-measurement-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xen</td>
<td>90598855d9da2a459a180d2836a178d42aa95e</td>
</tr>
<tr>
<td>Vmlinuz</td>
<td>d69d517c2d03d3013a382441b54d4ad7b343</td>
</tr>
<tr>
<td>Initrd</td>
<td>e249e20581c80b84db66e4dd4d396bcole23acc6</td>
</tr>
<tr>
<td>Libc</td>
<td>c25e50330900c42182a59e79d79026c9203c97c9</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

B. evolution procedure of Trusted chain

Figure 7 illustrates the successful results of evolution the trusted chain on virtualization computing platforms. The detail steps as follows.

Firstly, we deploy the baseline-measurement-values of all programs which are loaded during the boot process of physical machine or virtual Machines on virtualization computing platform in the file named InitialConfig.xml.

Secondly, the TPRTM server starts running, and the TPRTM client also starts at the same time, the evolution procedure of trusted chain begins working which uses the algorithm of TPRTM. Figure 6 shows the successful result of updating program of xen kernel on trusted chain.

Thirdly, we assume xen is maliciously altered, so the baseline-measurement-value of xen has been changed from \( fc69c125fd66ca69cab96402ce2f10a501b306 \) to \( fc69c125fd66ca69cab96402ce2f10a501b307 \). Our system could prevent xen kernel from running continually. At last, warning information is printed on the screen.
B. Performance Evaluation

Our experiment results show that the average completion time of a TPM seal operation is 610.778 ms. And the average computation time of a Measurement is 13.701 ms. As shown in Figure 8, the verification mechanism on the virtualization platform proposed in this paper leads to around 2000 ms delay for the boot process. The reason for this delay is that in verification mode, six programs which belong to 3 Groups are measured and verified. The six programs are as follows: GRUB_STAGE1, GRUB_START, GRUB_STAGE1_5 and GRUB_STAGE2 belong to Group1, vmlinuz-2.6.32.12 and initrd-2.6.32.12 belong to Group2, libc.so and bash belong to Group3. The time consumption for measurement is 8*13.701 ms = 109.608 ms in total.

While the trusted chain is enabled, the programs in Group1 will be unsealed in Virtual BIOS which includes an unseal operation, the programs in Group2 will be unsealed in GRUB_STAGE2 which also includes an unseal operation, the programs in Group3 will be unsealed in kernel which as well as includes an unseal operation.

Consequently, time consumption for the three unseal operations is 3*680.778 ms = 1832.334 ms in total. So if the system is booted in attestation boot-mode, Time consumption increases 109.608 ms + 1832.334 ms = 1941.942 ms. The result is roughly the same as the experiment result shown in Figure 9 which horizontal-axis shows that the time consumption for attestation boot-mode during the boot process is around 2000 ms when the number of virtual machines is 10, 50 or 70. The vertical axis of Figure 9 shows that the construction of trusted chain on virtualization platform leads to the performance degradation 2.23% in average, which is acceptable for the security that it brings. Figure 10 gives the stereographic of time consumption of each stage of two boot-modes on platform for the sake of clearness.

V. CONCLUSION

In this paper, we present a method to construct a trusted chain on virtualization computing platform to avoid the problems of previous works, on-continuous chain, uneasy to update the chain and insecure to perform remote attestation. Meanwhile, we design an architecture and implement it as a prototype system. The prototype has been adopted as an integritys-ensuring mechanism in a large communication enterprise. The system is now running stable, and can construct, update and maintain the chain easily, securely and flexibly. Therefore, this work will help to lay a foundation for the key technology to build virtualization-based trusted cloud computing. The trusted virtualization platform has many issues to be researched and discussed, our future work will focus on monitoring the whole life cycle integrity of programs on virtualization computing platform.

ACKNOWLEDGMENT

We thank the anonymous reviewers and my supervisor Yuzhong Sun for his helpful suggestions. This work is supported by National Natural Science Foundation Commission of China under grant NO. 60921002.

REFERENCES
