Software Behavior Based Trustworthiness Attestation For Computing Platform

Peiqiang Chen
Century College, Beijing University of Posts and Telecommunications, Beijing, 102613, China
chenpq191@gmail.com

***Abstract*—With a prevalence of pervasive computing, especially cloud computing , the software is at the core and play a vital role. This advance the security problem, so software trust is drawing increasing attention. Therefore, we need a unified trust relationship model between entities, which captures both the needs of the traditional computing world and the world of pervasive computing where the continuum of trust is based on identity, physical context or a combination of both. Here, we presents a software behavior based attestation model which try to determine the trust state of attesting platform from its system trust related behaviors. The new attestation model has advantages of privacy protection and high feasibility. In addition, it can also help to control and limit the impacts of security accidents such as malicious code in system. This paper also proposes a trust framework for service oriented application and displays its formalization model. It is useful for designing trust and reliable system and helpful for software developer's analysis and validation of the application.**

***Index Terms*—Trusted formal definition, Trusted computing, Software behavior, Trusted attestation**

I. INTRODUCTION

The development of the industry depends on the What is trust? TCG (trusted computing group) gives the definition of trust as follow: Trust is the expectation that a device will behave in a particular manner for a specific purpose [1].

Trusted attestation is becoming a hot topic in information system security, and it is one of the most important content in trusted computing. Trust attestation often occurs in such a situation: when a user on platform A accesses resources in platform B, or when B try to dispatch a computing task to A, B will verify whether A is trustworthy besides the permission check against user. The process to measure and verify the trust state of a platform is the trustworthiness attestation of computing platform.

The traditional trusted attestation approaches are the configuration[2-4] based and property[2-3] based attestation. The reason that these two attestation approaches are impractical is that neither platform configuration nor system properties can directly and exactly represent the trustworthiness of a platform. In fact, the expectation of software behavior is generally expressed by system security policy, if the security policy of attesting platform conforms to the verifier's expectation, then we say that the attesting platform will behave as the verifier's expectation, that is, the attesting platform is trustworthy.

To make the platform attestation more practical, we here present a new attestation model which is based on the software trustworthiness related behaviors. Compared with the configuration based and property based attestation, software behavior based attestation has following advantages:

A. It concerns only software trustworthiness related behaviors, so the analysis is more exact, simple and efficient.

B. It protects the platform configuration privacy from others.

C. In addition, software behavior based attestation can also be used to effectively constrain impacts caused by malicious code such as Trojan and virus which are common in today's business systems.

II. RELATED CONCEPTION

In this chapter, we first explain some concept of this paper we used.

Trust is a relationship between two entities such that one entity believes, expects, and accepts that the other trusted entity will act or intend to act beneficially [5]. Trust represents the degree to which a entity would be trustworthy, secure, or reliable in any interaction with the other entities.

A security policy is responsible for assigning credentials to entities, delegating trust to third parties, and reasoning about users' access rights. The security policy is to state the required security specifications.

The verifier's expectation is what the verifier expects the result is caused by software behavior and what the verifier expects to software behavior or system behavior do.

III. ATTESTATION MECHANISM IN TCG

In TCG (Trusted Computing Group) technical specifications, attestation is the process of vouching for the accuracy of information, a platform can attest to its description of platform characteristics that affect the integrity(trustworthiness) of a platform.

Manuscript received Mar. 2, 2011; revised Apr. 3, 2011; accepted Apr. 10, 2011.

doi:10.4304/jsw.7.1.55-60

Core of trusted computing is a hardware chip (Trusted Platform Module, TPM) which mainly contains cryptographic units and secure storage units. TPM protects its stored objects in shielded location which can be accessed only by protected capabilities.

TPM keeps specific platform state information in a group of platform configuration registers (PCR), and each platform provides proof of trustworthiness by an attestation identity key (AIK) of TPM. PCR will be reset on system reboot and cannot be set to any particular value by anyone. PCR can only be extended by predefined events, for example, when a predefined event occurs, the platform will extend the event data digest into the responding PCR as: PCR[n]=hash(PCR[n]+hash(event data)), so PCR not only preserves the event related information but also the event history and its order.

When a challenger requests the proof of trustworthiness of an attesting platform, it also requests the responding PCR value from that platform. An agent on the attesting platform will collect the proof data and request the built-in TPM for AIK signed PCR and returns to the challenger as well as credentials that vouch for the TPM. The challenger verifies the proof and other information returned by the platform agent and determines the trustworthiness of the attesting platform.

The trustworthiness of platform agent can be assured by a process of transitive trust, where the trust boundary can be extended from the Root of Trust to a second group of functions, etc. System boots from Core Root of Trust Measurement (CRTM) to form a trust chain as CRTM->BIOS->OS Loader->OS->Application, and so does the platform the agent [6].

IV. TRUST OF SERVICE

The trust of a web service needs to be managed from service discovery to behavior monitoring. We classify the trust management of a service into four steps as follows:

Service discovery: find out appropriate services from service center. From service center, many function similar services can be discovered. The service center needs to be trusted.

Service selection: select the most proper service from result in precious phase according to the reputation, performance and so on.

Service composition: obtain a composite service with internal consistency. The interfaces of different services in a composite service must be consistent. The execution sequence must be arranged logically to avoid deadlock. This can ensure that the composite service is trusted.

Behavior monitoring: monitor the services'behavior. A service, which is considered to be trusted as to the trust degree model, may have bugs or unwanted operation while executing.

The different phases and corresponding resource are displayed in Fig. 1.

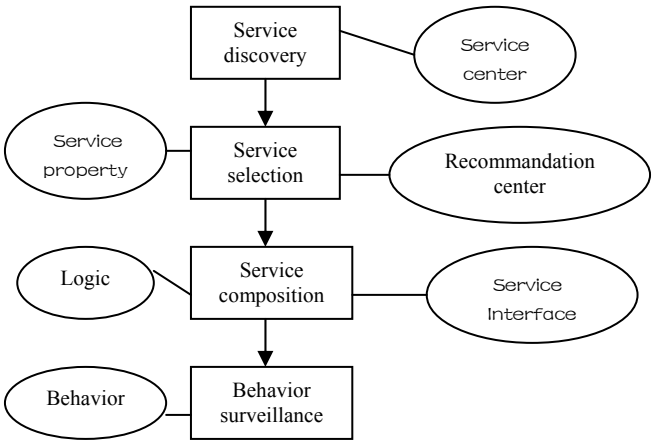

Figure 1. Trust management and resources in different phases

The service management in different phase relates to different resources. For example, in service selection phase the service property, such as functions and runtime environment information, needs to be considered to evaluate the trustworthiness of service.

V. THE FORMALIZATION MODEL

The formal modeling can be helpful to analyze the attributes of the system and to verify the properties.The advantage of formal method in software development is discussed in [7]. The system designer can comprehend the system clearly through the formal model. The light weight formal framework of service oriented application in [8] does not take trust into consideration. We define the formal models of service oriented environment and the application domain.

A. Trust degree model

The trust degree of a service relates the service description V_a, known part recommendation V_{kp} and unknown part recommendation V_{up}. The model can be defined as a function about the three factors, and that is:

Trustdegree(A, S) = f (V_{kp}, V_{up}, V_a)

The value of Trustdegree(A, S) represents the trust degree and it is between 0 and 1. Vkp and Vup are the third part's recommendation about service S based on the query of the trusting agent A and its value is between 0 and 1. They may be expressions about various factors, such as the authenticity of the description, the popularity and so on. The function f varies with different systems and can be defined by the software analyzer. For example, Trustdegree(A, S) =$0.3*V_{kp}$ +$0.1*V_{up}$ +$0.6*V_a$. This is a simple model and the service description possesses great proportion. In practice more factors will be taken into consideration, such as runtime factor.

In general, the value of Trustdegree could not be 1. We deem that the service is trusted if its Trustdegree is higher than a given border value. Because it is impossible to obtain totally-satisfying services, the monitor of the execution process is needed.

B. The process of trust management

There are lots of services, S_1, S_2, …, S_n, in an application. A service can be a simple service or a composite service. The techniques in [9-10] for service integration can be employed for our use. Composite service (CS) is consisted of several coordinated services. Services involved in the application must "coordinate" their work to solve the computational problem. Here, we mean "coordinate" as any kind of interaction between involved services that aims to achieve the application goal. Coordination may be implemented by different means, including but not limited to, exchanging data, exchanging messages, service provisioning, service control, service monitoring etc [11].

In order to manage the services simply, we integrate the closely-related services into a coarsegrained composite service:

$CS = <\{ S_1, S_2, \ldots, S_n\}, DO, In, Out, MA>$

Where S_1, S_2, …, S_n are the sub-services of CS and n indicates number of logical services, which can be an atomic or a composite service, deployed in the composite service. And that, CS has its own data objects (DO) and input (In) and output (Out) interfaces. MA represents monitoring agent.

In addition, a composite service corresponds to a monitoring agent about which is used to monitor the behaviors the composite service and deal with abnormities. A monitoring agent has some monitor conditions (Cond) and an Exception Handler (EH). If all the conditions are all true, the monitoring agent works silently. Otherwise, it implements the corresponding operations according to the false conditions. A monitoring agent corresponds to a composite service and they are related by the name of the composite service:

$MA = <Cond1 \wedge Cond2 \wedge \cdots \wedge Condm, EH>$

Therefore, a composite service and its corresponding monitoring agent constitute a subapplication which can perform certain functions and have fault-tolerant capability.

A service oriented application A is composed by several services and these services are interrelated by relationship R. An application can be expressed by:

$A = <\{S\}, R>$

Where, S may be a simple or a composite service, R denotes the relationships among the set of subapplications and it can be expressed by a graph G or natural/formal representations. These sub-applications can associate with each other by the data objects or interfaces.

All the services and the agents in the application are in the static environment. When the application runs, only some of them work.

C. Formal analysis

In the service elicitation phase, the trust degree model is employed to do the selection. The trust degree model is defined by the system analysts according to different domain characteristics.

The formal specification can be used to verify the correctness or integrity both in design and implementation phrase. In the system design phase, the formal verification can be used for three purposes. The first purpose is that for each output of any service a matched input must be existed. Second, the parameter of service needs to be consistent with its type. Third, data objects in composite service are consistent with its included simple service. The conditions of MA should match with the ones of corresponding ExceptionHandler. In implementation phrase, the conditions of monitoring agent can be checked at intervals. These conditions are expressed by Boolean expressions and they are easy to check.

Also the designer can see the functionalities and the associations of different entities from the formal specifications.Reasoning can be implemented to detect the inconsistency both in the system design and in composite services.

VI. SOFTWARE TRUSTED RELATED BEHAVIORS

As introduction, from the verifier's point of view, if the security policy of the attesting platform complies with its expectation, then the platform will behave in expectation, so we use system security policy to denote the trust state of attesting platform. On the other hand, system security policy can be only changed by software trustworthiness related behaviors which may alter the system security. We concern these software trustworthiness related behaviors because the measurement and analysis of them can help to get the state of the system security policy more real and more exactly than directly measuring the security policy.

For the sake of compatibility of policy, we presume the verifier and the attesting platform are in the same security domain.

Software behaviors can be defined as a quad-tuples b,

$b \in B = S \times P \times I \times O$, where:

$S=\{s_1, s_2, \ldots\}$ indicates the set of all subjects in the system;

$P=\{p_1, p_2, \ldots\}$ indicates the set of executable programs;

$I=\{i_1, i_2, \ldots\}$ indicates the set of behavior inputs;

$O=\{o_1, o_2, \ldots\}$ indicates the set of behavior results.

To a particular software behavior $b=(s, p, i, o) \in B$, we will use b.s, b.p, b.i and b.o to represent the subject, action, input and output of software behavior respectively. Moreover, we use R to represent the set of the software trustworthiness related behaviors where $R \subseteq B$, to any software trustworthiness unrelated behavior b, $b \in B$ and $b \notin R$, we can simply suppose it to not be trustworthy.

Then we give the definition of trustworthy software behavior as:

Definition 1: Software behavior $b=(s, p, i, o) \in B$ is trustworthy if the changes of platform security policy caused by software behavior b complies with the user's expectation.

Software behavior has another definition. It can be defined that the software subject applies a service to a software object, that we call a behavior. We can express that as b=s:f(o),where b is behavior, s is subject, f are operation, service or function, o is object, f(o) refers to f will act ,be acting or have acted on object.

We use B to represent the set of the software behavior. And we use SP to represent the set of the platform security policy, and use E to represent the set of the verifier's expectation. Then we give the definition of trustworthy software behavior as:

Definition 2: Software behavior $b \in B$ is trustworthy if the changes of set SP of platform security policy caused by software behavior b,which means b=s:f(o),complies with the set E of the verifier's expectation, which can be expressed $SP \subseteq E$.

We can also use R to represent the set of the software trustworthiness related behaviors where $R \subseteq B$, and to any software trustworthiness unrelated behavior b, $b \in B$ and $b \notin R$, we can simply suppose it to not be trustworthy. And if software behavior b is trustworthy, then $b \in R$.

Because of differences in role, verifiers might have expectations that are not exactly same with real security policy on attesting platform, e.g., the verifier may expect that user A on the attesting platform have a most permission of "read" and "write" to file f, if the administrator of the platform changes the permission to "read only", then the verifier will regard this behavior as trustworthy, or if the verifier's expectation to user A's permission on f is "read only", and the administrator of the attesting platform changes it to "read" and "write", then this behavior will be regarded as untrustworthy by the verifier even if the administrator considers it to be appropriate.

The trust state of attesting platform is affected by software behavior, so we use a transform function to describe this impact as $T(b, r_i)= r_{i+1}$, which indicates software behavior b makes the platform change from state r_i to state r_{i+1}. More generally, if $SB=b_1b_2\ldots b_n$ represents a sequence of software behavior, where b_1, $b_2,\ldots,b_n \in B$, then $T(SB, r_i)= r_{i+1}$ means a software behavior sequence SB makes the platform change from state r_i to state r_{i+1}. A trustworthy software behavior will not change the trustworthiness of platform, so we give the following theorem.

theorem 1: If attesting platform is trustworthy in state r_i, software behavior $b \in B$ is trustworthy, and $T(b, r_i)= r_{i+1}$, then the platform is trustworthy in state r_{i+1}.

Proof. If attesting platform is trustworthy in state r_i, software behavior $b \in B$ is trustworthy, according to definition 1 when software behavior b makes the platform change from state r_i to state r_{i+1}, the changes of set SP of platform security policy caused by software behavior b ,which means b=s:f(o),complies with the set E of the verifier's expectation, which can be expressed $SP \subseteq E$, so the software behavior b can not change the trustworthy state of platform, then the platform is trustworthy in state r_{i+1}.

Based on theorem 1, we can easily prove the following conclusion by induction.

Conclusion 1: If attesting platform is trustworthy in state r_i, and each b_j ($1 \leqslant j \leqslant n$) in the software behavior sequence $SB=b_1b_2\ldots b_n$ is trustworthy, then the platform is trustworthy in state r_{i+1}, where $r_{i+1}=T(SB, r_i)$.

Proof. If attesting platform is trustworthy in state r_i, and each b_j ($1 \leqslant j \leqslant n$) in the software behavior sequence $SB=b_1b_2\ldots b_n$ is trustworthy, according to theorem 1, so $T(b_1,r_i)$ is a trustworthy state, and $T(b_2,T(b_1,r_i))$ is a trustworthy state, which means $T(b_1b_2,r_i)$ is a trustworthy state. And then $T(b_3,T(b_1b_2,r_i))$ is also a trustworthy state, that is $T(b_1b_2b_3,r_i)$ is a trustworthy state…and so on. In the end, $T(b_n,T(b_1b_2\ldots b_{n-1},r_i))$ is a trustworthy state, that is $T(b_1b_2\ldots b_n,r_i)$ is a trustworthy state, so $T(SB,r_i)$ is a trustworthy state. We can express this state r_{i+1},so $r_{i+1}=T(SB,r_i)$.

The importance of conclusion 1 lies that we have a practicable way to determine the trustworthiness of attesting platform, because the software behavior is a specific and touchable definition which we can measure, report and verify.

As we said above, any software trustworthiness unrelated behavior will be regarded as trustworthy, so we can remove those behaviors from the SB in conclusion 1, thus we can greatly reduce the amount of software behaviors need to be measured and analyzed, and therefore increasing the computational feasibility.

Following is the main process of software behavior based attestation:

A. To prove or make assure the attesting platform is trustworthy in the initial state after system startup [12-13].

B. To verify or make assure that all software trustworthiness related behavior after system startup are trustworthy, so as to determine the trustworthiness of the platform.

VII. SOFTWARE BEHAVIOR BASED ATTESTATION MODEL

According to conclusion 1, we present a new attestation model which is based on the software trustworthiness related behaviors. It remove those software trustworthiness unrelated behavior from the SB in conclusion 1, and only need to prove or make assure the attesting platform is trustworthy in the initial state and to verify or make assure that all software trustworthiness related behavior after system startup are trustworthy, so as to determine the trustworthiness of the platform is in a trustworthy state based on conclusion 1.

It is verifier who determines whether the attesting platform is trustworthy based on the trustworthiness measurement report. Fig.2 is the software behavior based attestation model. In Fig.2,the software behavior based attestation model is supposed to be composed by the three main components of verifier : A. security policy expectation, which is composed of expectation items; B. software trustworthiness related behavior verification module, by which verifier receive report from attesting platforms and verify the trustworthiness of the attesting platforms against the security policy expectation; C. the maintenance module for security policy expectation, which provide a mechanism to add, delete, modify items in security policy expectation. The verification process includes: A. integrity or authenticity verification to the action (program) of each software trustworthiness related

behavior(conclusion 1), B. analysis of result for each software trustworthiness related behavior (conclusion 1), the purpose is to check whether the software trustworthiness related behavior(conclusion 1) conflict with the defined security policy expectation.

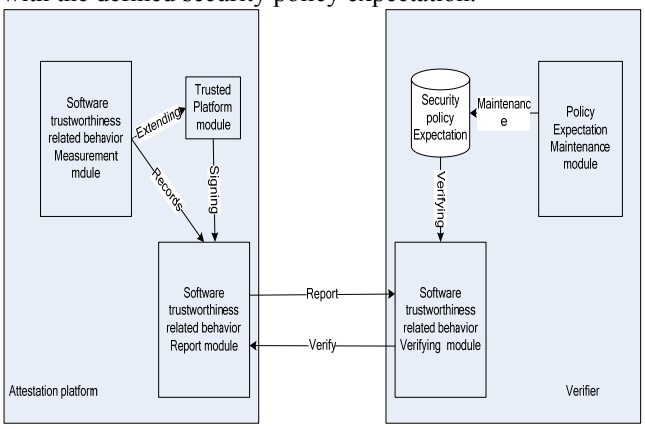

Figure 2. software behavior based attestation model

In Fig.2,The three main parts on attesting platform are software trustworthiness related behavior measurement module , software trustworthiness related behavior report module and TPM(Trusted Platform Module). The first is responsible for the event data recording and extending which are limited to the software trustworthiness related behavior(conclusion 1).In Linux, it can be implemented by LSM (Linux Security Modules) similar with the enforcer [14]; The second is responsible for responding the requesting from the verifier and report the event records to it. Both can be regarded as part of OS and their trustworthiness can be assured by the same way with OS. The TPM is core of trusted computing which mainly contains cryptographic units and secure storage units. TPM protects its stored objects in shielded location which can be accessed only by protected capabilities. TPM keeps specific platform state information in a group of platform configuration registers (PCR), and each platform provides proof of trustworthiness by an attestation identity key (AIK) of TPM. TPM assures integrity and authenticity of software trustworthiness related behavior(conclusion 1) records by signing the responding PCR value.

To determine whether software behavior of attesting platform complies with verifier's security policy expectation, a constraint of rule conformance should be added to each expectation item, e.g., an item in verifier's security policy expectation says that user A on the attesting platform have a permission no more than "read" and "write" to file f, where "no more than" is actually a constraint. These constraints are necessary because the verifier's security policy expectation is not necessarily exactly same with the platform security policy, as we discussed above.

The verifier checks the system trustworthiness related behavior of each attesting platform against its security policy expectation, e.g., a record of (root, install, "/bin/antivirus", success) shows that root user has successfully installed the anti-virus software from the attesting platform, but it is not supported in verifier's security policy expectation, so it is untrustworthy. Again, a record of (root, del, "Li Fang, /etc/fileA", success) indicates the root user has successfully deleted Li Fang's fileA and this behavior is trusted if the user's expectation is that.

Software behavior based attestation has impact on performance of attesting platform and verifier as well. The main additional cost brought to attesting platform is the processes that record software behavior event data. The events to be recorded are limited to the software trustworthiness related behavior, so the time cost can be ignored. To verifier, the performance depends on the number of the received software behaviors and the size of the security policy expectation. The main challenge to the verifier may be the cost to analyze the result of the software behaviors, that is, the impact that the software behaviors have on the trust state of attesting platform. But with the experience of any unknown thing is untrustworthy, any unknown software behaviors can be regarded as untrustworthy, so the order of complexity of the above analysis will greatly reduce.

VIII. RELATED WORKS

Platform trustworthiness attestation is an important research in trusted computing, An attestation approach is configuration based attestation in which the specific hardware/software configuration of the attesting platform is measured, and the trust state of the attesting platform is determined on the basis of the reported configuration information. Configuration based attestation is a simple approach, but it has a few deficiencies, for instance, leakage of platform configuration privacy and bad scalability[2-4]. Another type of attestation is property based attestation[2-3], but property itself is an abstract concept, it is hard to define or determine which property is related with or unrelated with the trustworthiness of attesting platform in a real system. Besides the related work introduced, R.Sailer etc. presented the Integrity Measurement Architecture (IMA) based on the TCG technical specification and implemented IMA on Linux platform [17]. A.Seshadri etc. try to determine the trustworthiness of attesting platform by discovering the time difference in accessing a random memory between a most optimized software and a tampered one [18]; V. Haldar etc. try to attest the remote platform by program semantic analysis which claims to be able to avoid the deficiencies of coarse grain and static in some attestation models [19].

IX. CONCLUSIONS

This paper studied a software behavior based attestation model which tries to determine the trust state of attesting platform from its system trustworthiness related behaviors. Software behavior based attestation determines the trustworthiness of attesting platform by its trustworthiness related behaviors. Compared with the other attesting approaches, it concerns only platform trustworthiness directly related factors; therefore it has

advantages in platform configuration privacy protection and better feasibility.

In this paper we also propose a trust framework for designing trustworthy service oriented applications, including service selection and service behavior monitoring. We use quantitative trust degree model for service selection and monitoring agents for the behavior surveillance. The formalized model of the service oriented application is also described. By using the formalization the formal verification can be performed.

In future, we will continue make more research in improving the verifier's performance to verify the software behaviors in a large and complex system, in which each user terminal has different security assumption. For formal analysis future work will concentrate on the reasoning rules in the formal model for formal analysis.

ACKNOWLEDGMENT

The authors wish to thank Jørgen Bøegh, Yuan Yuyu. This work was supported in part from them.

REFERENCES

[1] TCG. TCG Specification Architecture Overview,Version1.4. http://www.trustedcomputinggroup.org/

[2] Ahmad-Reza Sadeghi, Christian Stüble. Property-based Attestation for Computing Platforms: Caring about prop erties, not mechanisms. New Security Paradigms Works hop, September 2004.

[3] J. Poritz, M. Schunter, E.V. Herreweghen, and M. Waidner.Property attestation — scalable and privacy-friendly security assessment of peer computers, IBM Research Report RZ 3548, 2004. http://domino.watson.ibm.com/library/cyberdig.nsf/papers/215E33C2B4F7FA485256E97002A0D6C/$File/rz3548.pdf.

[4] E. Shi, A. Perrig, and L. van Doorn. Bind: A fine-grained attestation service for secure distributed systems. In Proc. of the IEEE Symposium on Security and Privacy, pages 154--168, 2005.

[5] D.H. McKnight, N.L. Chervany, The Meanings of Trust, Technical Report,University of Minnesota, 1996.

[6] TCG. TCG Specification Architecture Overview,Version1.2. https://www.trustedApplicationgroup.org.

[7] Andrews, D. "Formal methods and software development", Software Engineering: Education and Practice, IEEE Computer Society Press, 1996, pp.106-113.

[8] Aliaksei Yanchuk, Alexander Ivanyukovich, Maurizio Marchese, "A Lightweight Formal Framework for Service-Oriented Applications Design" ICSOC 2005 LNCS 3826, 2005, pp. 545-551.

[9] J. Zhang. "Trustworthy web services: Actions for now", IT Professional, Vol. 7, No. 1, 2005, pp. 32-36.

[10] Sungkeun Park, Ling Liu, Calton Pu, Mudhakar Srivatsa, Jianjun Zhang, "Resilient Trust Management for Web Service Integration", ICWS 2005, pp.499-506.

[11] Adrian Baldwin, Simon Shiu, Marco Casassa Mont, " Trust Services: A Framework for Service-Based Solutions", COMPSAC 2002, pp.507-513.

[12] William A. Arbaugh, David J. Farber & Jonathan M. Smith: A Secure and Reliable Bootstrap Architecture, in:Proceedings of the 1997 IEEE Symposium on Security and Privacy, pp. 65-71, May 1997.

[13] Huang Tao, Shen Chang-Xiang. A Trusted Bootstrap Scenario Based Trusted Server, Journal of Wuhan University (Nat. Sci. Ed.), Oct. 2004,Vol.50 No.S1:12-14.

[14] J. Marchesini, S.W. Smith, O. Wild, J. Stabiner, and A.Barsamian. Open-source applications of TCPA hardware. In Applied Computer Security Applications Conference, 2004.

[15] Daniel A. Menascé, Security Performance, IEEE Internet Computing,Vol.7,no.3,2003:84-87.

[16] Amtel. Trusted Platform Module AT97SC3201 Summary. http://www.atmel.com/atmel/acrobat/2015s.pdf.

[17] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In Proceedings of USENIX Security Symposium, pages 223–238, Aug. 2004.

[18] A.Seshadri, A.Perrig, L. van Doorn, and P. Khosla. SWAtt:SoftWare-based Attestation for embedded devices. In Proceedings of IEEE Symposium on Security and Privacy, May 2004.

[19] V. Haldar, D. Chandra, and M. Franz. Semantic remote attestation - a virtual machine directed approach to trusted computing. In Proc. of the Third virtual Machine Research and Technology Symposium. USENIX, 2004.

Peiqiang Chen Beijing, China. Birthdate: Oct, 1969. is Computer Software and Theory Master, graduated from College of information science and technology Beijing Normal University. And research interests on information safety and trusted computing.

He is a senior lecturer of Dept. Computer Science and Technology Century College Beijing University of Posts and Telecoms.