http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (10): 1905-1913, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.1905.1913

A User-specific Trusted Virtual Environment for Cloud Computing

He Rongyu, Wu Shaojie and Jiang Lu
Zhengzhou Information Science and Technology Institute, Zhengzhou, 450004, China

Abstract: Cloud computing is a new computing model that provides users multiple isolated execution environments, usually referred to as Virtual Machine (VMs), on a single host. But the model that all users' data place in the cloud brings a significant level of risk on the privacy, for example, the data in the cloud could be tampered or abused by malicious customers. So, the trustworthy of the VMs becomes a significant hurdle for mainstream cloud adoption, especially for critical or sensitive applications. In this study, we apply Trusted Computing technology into the Cloud computing infrastructure, design a multi-source trust chain model, a Tree-like Trust Chain for VMs (TTCVM), which provides VMs with two trust sources, trust from the host and trust from the user. A user-specific virtual TPM, μTPM, is proposed using the TPM virtualization which aggregates the two trusts and forwards them to the user VM. Formal analysis demonstrates that TTCVM provides users the visibility and configurability into security settings of his execution environment.

Key words: Cloud computing, trusted computing, trust transitive, virtualization

INTRODUCTION

Cloud computing is considered as a new consumption and delivery model for IT services (Michael, 2010). The cloud delivers massively scalable computing resources as a service using Internet technologies which allows these computational resources to be shared among a vast number of consumers with a lower cost of IT ownership (Anitha and Keerthana, 2013). So, the cloud has many benefits, such as lower overall cost of IT ownership, increased flexibility, fault tolerance, locality flexibility ability and to respond to new business requirements quickly and efficiently (CSA, 2009).

Cloud computing simulates multiple isolated execution environments on a single computer using virtualization technology and the isolated execution environments is often referred to as Virtual Machines (VMs). User rents one or more VMs from the Cloud Computing Providers (CCPs) to run his operating system and applications and this cloud delivery model is known as Infrastructure as a Service (IaaS). This new computing model brings new risks and challenges on the privacy. In the IaaS model, the control of the host computing platform is taken from users themselves to the CCP and the user loses their domination to security settings on the host in physically. Furthermore, the Cloud computing is a multi-tenancy platform and it services for many users simultaneously. So, the privacy and security of information on the platform have become the most attention problem for users. For example, many organizations such as financial institutions, health care providers and government agencies are legally required to protect their data from compromise due to the sensitivity of their information. So, in order to protecting their data, these organizations need some rigorous physical and logical protection mechanisms to manage and maintain their own data centers (Krautheim, 2010).

The security and integrity of the VMs in the cloud is becoming a serious problem when an organization adopts cloud solutions (Duc and Keryell, 2005). In this study, we wish to place a level of trust in the VMs that provides a verifiably security execution environment for users. Trusted Computing (Miller and Pegah, 2007; Achemlal *et al*., 2011; Peinado *et al*., 2004) is one technology we can leverage to achieve our goals. Using a hardware-based trust anchor, Trusted Platform Module (TPM), the Trusted Computing can provide some protection mechanisms to protect the execute environment from compromise, for example, the Integrity measurements and reporting function can be used to prove the trustworthiness of the host platform. However, the Trusted Computing is aimed to solve the security issue for a single system, such as a Personal Computer. So, it couldn't provide fine-grained protection mechanisms for the Cloud, such as individual VMs.

In this study, we design fine-grained trusted model for the Cloud by expanding the trust chain in existing TCG which provides trusted service to multiple VMs for

Corresponding Author: He Rongyu, Zhengzhou Information Science and Technology Institute, Zhengzhou, 450004, China

various users. We propose a user-specific virtual TPM by TPM virtualization and design a Tree-like Trust Chain model for VMs. The model can provide user the visibility and configurability into security settings of his VM.

RELATED WORKS

The granularity of the rental resources in a Cloud computing platform is VMs, so the CCPs prove the integrity of VMs will improve acceptance of his cloud platform. Users understand the configuration and the security settings (such as hardware and software stack) through the VM integrity metrics and then determine the trustworthy of the platform.

Trusted Computing (Loscocco and Wilson, 2007) is the most direct method of platform integrity protection and it has been permeated into the Cloud platform. IBM has developed an Integrity Measurement Architecture (IMA) (Sailer *et al*., 2004) which measures all executable components on the platform before it execution and stores these measurement value into TPM as Reference Integrity Measurement (RIMs). When the user starts his virtual machine, the IMA provides the validation and attestation of the host using the RIMs. But the IMA only guarantee the integrity of the host platform, it can't guarantee the integrity of each VM.

The TPM virtualization could provide a virtual TPM for each VM in the Cloud platform and it provides enhanced security by binding the virtual TPMs to the physical TPM. Using the TPM virtualization and the IBM4758 cryptographic coprocessor, Berger *et al*. (2006) implement a virtual TPM, named vTPMs; Based on the vTPM, Scarlata *et al*. (2008) established a TPM virtualization framework through software emulation. It creates a virtual TPM for each virtual machine and connects the virtual TPM to physical TPM by a certificate chain. However, since the vTPM is opaque for the operating system, it is difficult to implement.

Private Virtual Infrastructure (Krautheim, 2010) represents a new management and security model for cloud platform where a special virtual machine appliance, LoBot, is used to locate trustworthy platforms. Santos *et al*. (2009) focuses on building trusted cloud computing platforms using the TCG's specifications and establishes an architecture for Could Computing, named Trusted Cloud Computing Platform (TCCP). Before the providing cloud service, TCCP validates the integrity of the computing nodes using the Trusted Computing technology, then provides user VM an abstract closed execution environment. TCCP, so far, is only a conceptual model, there is no a prototype system available for testing. In addition, the resources in physical TPM are limited for each virtual TPM shared or replicated.

The Virtual Machine Monitor (VMM, also called a Hypervisor) also can provide integrity protection for cloud platforms in the VMs level. Terra (Garfinkel *et al*., 2003) provides a flexible trusted computing structure. It leverages the VMM as a trusted computing base for platform, creates isolated tamper resistant customer VMs on a host and provides trusted certification between user and application by the technology such as secure boot and validation. In the Terra platform, "Open Box VM" used to run a general purpose commodity OS (such as Windows or Linux) together with its typical set of applications while "Close Box VM" used to run security sensitive applications where the software stack is tailored to the security requirements of these applications. However, because there isn't any mechanism to measure the environment before the provisioning of the virtual machine, Terra cannot guarantee a secure launch of the virtual environment. Livewire (Garfinkel and Rosenblum, 2003), VMwatcher (Jiang *et al*., 2007) Antfarm (Jones *et al*., 2006) and XenAccess (Payne *et al*., 2007), etc., also use this method to provide users with integrity protection of virtual machines. But this method depends on the security services provided by the Hypervisor while Wojtczuk and Rutkowska (2008), Garfnkel *et al*. (2003) and Citrix (2012) studying shows that the Hypervisor subjected to security attacks and may be compromised.

A TREE-LIKE TRUST CHAIN FOR VM

The Tree-like Trusted Chain for VM (TTCVM) is a trust model for Cloud platform as Fig. 1 shown. It can provide trust service for multiple VMs on a single host, for various users, by one physical TPM.

In the TTCVM model, we introduce a new component, vTPM_Creator, into the traditional Trust Chain described in TCG specifications (TCG, 2003). The vTPM_Creator is responsible for merging the trust from user and that from the host platform and forwarding the merged trust to a user's VM. It is a privileged virtual machine with strong isolation and its integrity is guaranteed by the physical TPM. The vTPM_Creator is a software component with the functions of creating and destroying a virtual TPM, named as μTPM, for users. The μTPM is a user-specific security component which implements the full TPM specification. In addition, the μTPM can be configured by the user through the vTPM_Creator and provides the TPM service according to the user's security settings.

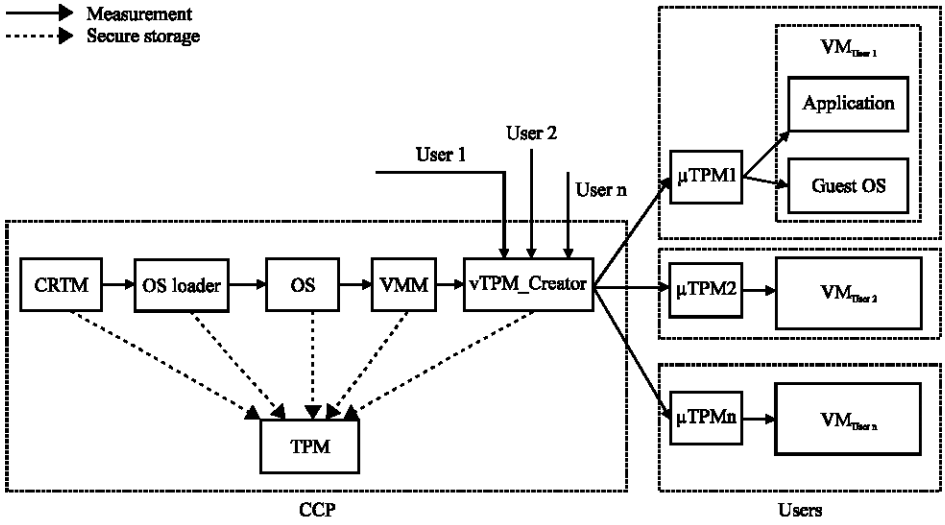

Fig. 1: Tree-like trust chain model

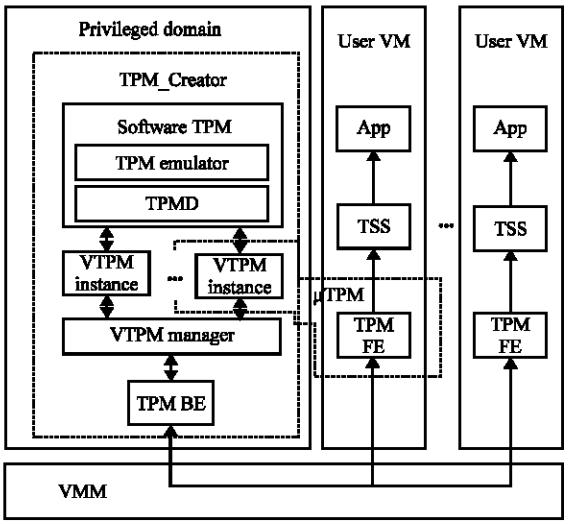

Fig. 2: Architecture of vTPM_Creator

Architecture of vTPM_creator: The vTPM_Creator receives the security police from users and creates a μTPM for his VM, the μTPM guaranties that the CCP will provide a trusted VM which satisfies the user's security requirement. The architecture of vTPM_Creator is shown in Fig. 2.

The vTPM_Creator is a software component running on the privileged domain (Dom0). It is composed of a TPM emulator, a vTPM Manager and a set of VTPM instance. The TPM emulator, named STPM, is bound to a physical TPM and emulates it to provide trusted service such as integrity measurement, cryptographic operations, etc. The trustworthy of the STPM is guaranteed by the physical TPM. The vTPM Manager is responsible for creating a μTPM according to the user's security polices, so the trust from user is integrated into the cloud platform. It is also responsible for destroying the μTPM when the user logout.

A VTPM instance is an isolated storage for a user and it is used to store the user's security settings for his VM. All the VTPM instance are stored in a shield storage area which can be accessed by STPM only.

User-specific virtual TPM -μTPM: The cloud provides an isolated computing environment for each user and the isolated computing environment is known as user VM. In our architecture, each user VM needs a TPM functionality to guarantee that the user VM satisfies the user's security requirement. Here we introduce a user configurability virtual TPM model for each user VM. The virtual TPM is named as user-specific TPM, represented as μTPM, because it can be configured by a user.

The μTPM is composed of a TPM Front End (TPM FE), a TPM Back End (TPM BE) and a VTPM instance, as Fig. 2 shown. vTPM_Creator creates a VTPM instance for each user VM according to user's security requirement, such as software whitelist, network access control *et al*. The user VM communicates with the μTPM using a split device-driver model where a client-side driver (TPM FE) runs inside the user VM that receives the TPM requisitions from user VM, appends the VM identifier to the requisitions and forwards it to the vTPM_Creator, it also picks up the responses and returns them to the user

VM. The server-side driver (TPM BE), runs in the vTPM_Creator which transmits the TPM requisitions from TPM BE to VTPM manager where the requisition is bound to his VTPM instance, the TPM BE is also responsible for forwarding the response back to the TPM FE according to the VTPM identifier.

FORMAL ANALYSIS FOR THE TTCVM

In the trusted computing platform, the TPM is a tamper-resistant device which is implicitly trusted by the user. Therefore, the TPM is usually used as a trust anchor to ensure the integrity of other hardware/software components by securely measurement and reporting. The Core Root of Trust for Measurements (CRTM) is the first code entity which runs on a platform providing the root for trust establishment on other hardware pieces and application layers. So, the trust is propagated from the CRTM to the BIOS to MBR to OS and finally to applications from the TPM.

A non-interference model based analysis method for trust propagation: Since, Goguen and Meseguer (1982) proposed a Non-interference model for information flow in 1982, there are many security model about information flow without interference were proposed in succession, such as Non-deducibility Model, Generalized Non-interference model and the Non-interference model is widely used as a tool for analyzing and certifying the security of an information system. Based on the Non-interference model, Zhang *et al*. (2009, 2010) proposed a formal method for analyzing the trust chain of trusted computing platform. The method is described as follows.

Definition 1: System M is composed of four collections: system state set S, action set A, output set O and security domain D, as well as four functions defined on these collections and can be expression as:

$$M = (S, O, D, A)$$

Here, S is a set of system States and the initial state s_0, $s_0 \in S$; A is a set of Action in the system, it includes the control operations come from the system itself and the input operations come from the outside; D is a collection of isolated Domains in the system. A subject is restricted in an isolated domain and it can observe the results of his control operations. The system is divided into a collection of isolated domains can restrict the information to be interfered in the flow; O is a collection of system Output.

Definition 2: Output function, $S \times A^* \neg O$, represents that if an operational, or a sequence of operations $a^* = a_1 a_2 \cdots a_n$ is executed, the system state will be changed from one state to another. It is denoted as:

$$s_j = O(s_i, a),\ a \in A;\ s_j \in S,$$

or:

$$s_j = O(s_i, a^*),\ a^* = \{a_1, a_2, \cdots, a_n | n>1,\ a_i \in A\};si,\ s_j \in S$$

Definition 3: The state of the system can be represents by a group of objects and its values. Rushby (1992) represents the state of the system by a group of Sets and Functions:

N = A countable set of name of objects, n(n∈N) is the name of an object
V = A countable set of value of objects, v(v∈V) is the value of an object

- The function: v = contents (s, n) represents that the value of an object named n is V when the system in the state S
- The function: P(n) = alter (u, s) represents that the set of objects whose value can be changed by domain u when the system in the state S
- The function: P(n) = observe(u, s) represents that the set of objects whose value can be read by domain u when the system in the state S
- The function: s' = step(s, a) represents that the next state s' of the system when action a is applied in state S

Definition 4: Binary relation ~> represents that an interference is happened between two information flows inside different domains and it is denoted as interference relationship. ↛ represents the complementary set of the interference relationship, named as non-interference relationship.

Corollary 1: If the relation:

$$\forall A, B \subset D;\ a, b \in A \cup c \in B;\ a \rightsquigarrow b \cup b \rightsquigarrow c \Rightarrow a \not\rightsquigarrow c \qquad (1)$$

is hold, then there is no unexpected interference in the system. So, we can say that a binary relation on D×D that satisfies the relation (1) is an Intransitive Non-interference relationship, it is denoted as $A \xrightarrow{IN} B$.

Theorem 1(Zhang *et al.*, 2010): The information flow in a system is an Intransitive Non-interference relationship if it satisfies one of the following four conditions:

- The domains in the system satisfy the output consistency. This means that the output of an operation only affects the domain which conducts the operation
- The action conducts in the system will change the state of the system and the change is only associated with the previous state of the domain which conducts the action
- If an action to change the value of an object, the domain conducting the action must has the 'write' right to access to the object
- Any two domains in the system satisfy the following relation:

$$\exists n \in N, n \in alter(u, s) \wedge n \in observe(v, s) \rightarrow u \rightsquigarrow v$$

The proof of the theorem can be found in Zhang *et al*. (2010).

Formal analyzing for TTCVM: The TTCVM model can be formal described as following: the system C is a triple, C = (O, D, A).

Here, C is the cloud platform which is composed of a collection of components. The component can be a piece firmware, such as BIOS, as well as software, such as the OS, a driver or an application. The components are denoted as s_0, s_1,$\cdots s_n$ and the s_0 is the first component that the cloud platform is booting.

O is a set of Outputs in the system.

A is a set of the Actions in the system, i.e., the control operation of the system itself. We define two actions in our architecture, Digest and Merge. The Digest represents checksum computing on and the Merge represents the merge computing.

D is a set of isolated Domains, the isolated domain can be considered as the components in the cloud platform. So, D = $\{s_0, s_1,\cdots,s_m\} \subset C$, $m \leq n$, i.e., the D is a collection of components that relates to the Trust transitive. In the TTCVM model, we have:

$$D = \{CRTM, OSLoader, OS, VMM, vTPM\text{-}creator, \{Udi\}, \{\mu TPM_i\}\}$$

where, UD_i represents a trusted domain for user U_i; μTPM_i represents the virtual TPM associated with user U_i.

The formal analysis for trust propagation in the TTCVM model is divided into two phases.

In the first phase, the trust is propagated along the trust chain:

$$CRTM \rightarrow OSLoader \rightarrow OS \rightarrow VMM \quad (2)$$

The components in the trust chain (A) are physical components and share trust by physical TPM. The trust propagates as follows:

Step 1: CRTM→OSLoader is trusted.

S_0 represents the root of trust, CRTM, of the system. The CRTM is the firmware of the system and which guarantee the trustworthiness of itself. So, we have:

$$s_1 = O(s_0, CRTM) \in TS \quad (3)$$

$$Digest(CRTM, OSLoader) = expect(OSLoader) \cup CRTM \nrightarrow OSLoader \Rightarrow CRTM \rightarrow OSLoader$$

Then we get:

$$s_2 = O(s_1, BIOS) \in TS \quad (4)$$

Step 2: OSLoader→OS is trusted.

In this step, the only operation of the system is digest operation (Digest) which is measurement operation in Trusted Computing. So, the interference between the components, OSLoader and OS, is satisfied Theorem 1(2), so, we get:

$$Digest(OSLoader, OS) = expect(OS) \cup OSLoader \nrightarrow OS \Rightarrow OSLoader \rightarrow OS$$

And then we can say:

$$s_3 = O(s_2, OS) \in TS \quad (5)$$

Step 3: OS→VMM is trusted.

According to the definition of the trusted computing, the interference between the OS and VMM is satisfied Theorem 1(2), so we can get:

$$Digest(OS, VMM) = expect(VMM) \cup OS \nrightarrow VMM \Rightarrow OS \rightarrow VMM$$

Then, the VMM is trusted:

$$s_4 = O(s_3, VMM) \in TS \quad (6)$$

In the second phase, the trustworthiness is propagated along the trust chain:

$$\{VMM, UD_i\} \rightarrow vTPM\_creator \rightarrow \mu TPM_i \quad (7)$$

Step 1: $\{VMM, Ud_i\} \rightarrow vTPM\_creator$ is trusted.

The trust from user and that from TPM are merged on the vTPM_creator and we get a hybrid trust by the Merge operation. So, the user UD_i will interference the virtual domain, s_4, of the platform, i.e.:

$$S_5 = S_4 \xrightarrow{UD_i} step(S_4, vTPM\_Creator) \quad (8)$$

According to the definition of our trust model, the user security settings (uSS) for his VM is shared in the vTPM_Creator, so for user UD_i, we have:

$$uSS \in alter(UDi, s4) \wedge uSS \in observe(vTPM\_creator, s_4) \rightarrow UDi \rightsquigarrow vTMP\_creator \quad (9)$$

This satisfies Theorem 1(4) and we get:

$$s_5 = O(s_4\{VMM, UD_i\}) \in TS \quad (10)$$

Step 2: vTPM is trusted.

This step is similar to traditional trust chain and we get:

$$s_6 = \mu TPM = O(s_5, vTPM\_creator/UDi) \in TS \quad (11)$$

From Eq. 2-11, we can prove that the trust model TTCVM is an Intransitive Non-interference relationship, so the trust chain in the TTCVM model is trusted.

IMPLEMENTATION OF THE μTPM

The μTPM, in which some user's special security policy is loaded, provides personalized security services for his VM, for example, it verifies the trustworthiness of the user's application, provides secure boost for the VM. The implementation of the μTPM is the process of the TPM Emulator being instantiated which includes the following two tasks:

Receiving the user's security policy and creating a μTPM: In order to transferring the user's security policy to his VM, a class, named TPM Control Structure (TPMCS), is defined in the vTPM_Creator, as shown in Fig. 3. When vTPM_Creator receives the security policy from user terminal through the trusted interface, it constructs a TPMCS object by instantiating the TPMCS class with the received user security policy and then a μTPM is established. The TPMCS object can be accessed only by vTPM_Creator for the security season.

The registers PCR [0...8] is used to save the values corresponding to that of the physical TPM's which include integrity measurement values of the BIOS, the GRUB, the BootLoader, the VMM and the kernel of the privileged domain. This collection of register values forms the immutable snapshots of the Cloud platform, it can be used as proof of trustworthiness of the Cloud and is read-only for vTPM_Creator; PCR [9...23] used to store the integrity measurement values of the user virtual machine, as well as the special security settings from the user, such as the static or dynamic integrity measure for critical applications and these PCRs are accessible for the μTPM. Therefore, the trust comes from the user and that comes from the Cloud Platorm are fusing in the μTPM.

Fields of the TPM control structure	
PCRs[0...8]	//TCB mesurement value
PCR[9...23]	//μVM mesurement value
Storage root key (SRK)	//Root of storage
Attestation identity keys (AIK)	
Endorsement key (E)	
Endorsement credintial	
M[i] [j]	//Reference values
	.
	.
	.

Fig. 3: TPM Control Structure diagram

When the user virtual machine is initiated in the authenticated boot model, the PCR [9 ... 23] is used to save the Integrity Measurement Values of components which is loaded in the initiation process and reports the integrity of the virtual machine to user; when the user virtual machine is initiated in the secure boot model, the PCR [9...23] is used to store the reference integrity measurement values of the components that will be loaded in the initiation process.

The PCR values is set in the process of a μTPM being constructed, shown in Fig. 4 and the details are depicted in Fig. 5. When the STPM is in its initialization process, it reads the data from PCRs of the physical TPM and copies the data into the correspondingly PCRs of the μTPM as its initial values. For a user's virtual machine, all the access requests to the TPM are redirected to the corresponding μTPM. Therefore, this accessing is independent and it doesn't interfere other virtual machines as well as the TPM_Creator. Meanwhile, this method can improve the efficiency of TPM service because the PCR data being maintained in the memory not only reduce the time consumed than that being maintained in physical TPM, but also reduces the workload of it.

µTPM working process: A TPM Emulator running in privileged domain provides TPM functionalities for user VMs and each user VM associated with a VTPM Instance which maintains the security polices and data. The VTPM manager is used to manage the VTPM Instances. The User VM communicates to the VTPM Instance through a Device Driver Module (DDM), which is composed of two modules, the DDM front-end (TPM FE) residing in the user virtual machine and the DDM back-end (TPM BE) locating in the privileged domain.

In the VTPM Instance, a 4-Byte identifier was used to identify the corresponding user VM. The VTPM manager builds a "VM-VTPM Instance" table to maintain the corresponding relation between the VTPM Instance and its VM.

When a user requests a µTPM, the VTPM manager will receive a request from the TPM_Creator, then it creates a VTPM Instance for the user by instantiating a TPMCS, forwards the user's security policy to the VTPM Instance. Finally, a user VM's identifier is assigned to the VTPM Instance and the µTPM is created. The TPM service requests from a user VM can be identified by the identifier and forward to his µTPM. The process of a µTPM working is shown in Fig. 6.

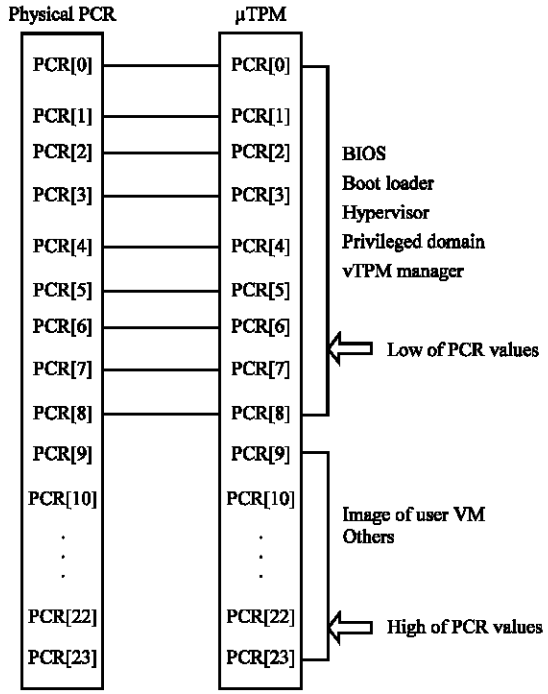

Fig. 4: µTPM PCR initialization process

- **Step 1:** User virtual machine sends a TPM Request to the TPM FE
- **Step 2:** The TPM FE forwards the Request to the TPM BE
- **Step 3:** TPM BE listens the Request, attaches user VM identifier into it and forwards the Request to VTM manager's Req_Queue
- **Step 4:** VTPM Manager will get a Request from the Req_Queue, extracts the identifier from the Request and then forwards it to the corresponding µTPM

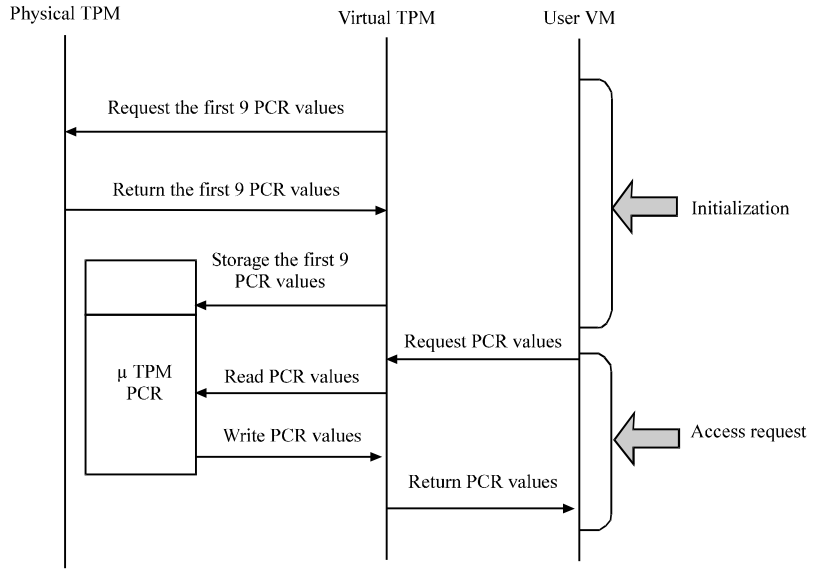

Fig. 5: Initialization of the specific process of PCR

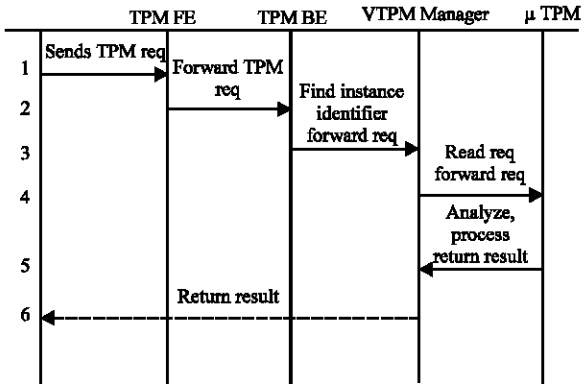

Fig. 6: µTPM implementation process

- **Step 5:** The µTPM processes the Request, generates the Result by combining the results and the identifier and returns the Result to the Ack_Queue of the VTPM Manager
- **Step 6:** VTPM Manager reads Result from the Ack_queue, forwards the Result to the corresponding user VM by the identifier through the TPM BE

CONCLUSION

Cloud Computing which delivers massively scalable computing and storage resources as a service with a low cost, caused a revolutionary change for information technology acquisition and service model. However, trustworthiness problem inherently in the Cloud Computing hinders it widespread adoption because costumers will loss control over their data in cloud computing and do not know where exactly their data is stored and processed.

In this study, we applied some TCG solutions into Cloud computing to building a trusted cloud blocks. We firstly proposed a multi-source trusted model, Tree-like Trusted Chain for VM (TTCVM), for the cloud computing infrastructure. Based on the TTCVM, a granular trusted model for the Cloud is designed which provides users the configurability and visibility, in the security, to his VMs.

REFERENCES

Achemlal, M., S. Gharout and C. Gaber, 2011. Trusted platform module as an enabler for security in cloud computing. Proceedings of the Conference on Network and Information Systems Security, May 18-21, 2011, La Rochelle, France, pp: 1-6.

Anitha, V. and N. Keerthana, 2013. An overview: Security in virtualization technology. J. Artif. Intell., 6: 112-116.

Berger, S., R. Caceres, K.A. Goldman, R. Perez, R. Sailer and L. van Doorn, 2006. vTPM: Virtualizing the trusted platform module. Proceedings of the 15th USENIX Security Symposium, July 31-August 4, 2006, Vancouver, Canada.

CSA, 2009. Security guidance for critical areas of focus in cloud computing. Cloud Security Alliance Guidance Version 3.0. http://www.cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf

Citrix, 2012. Vulnerability in xenserver could result in privilege escalation and arbitrary code execution. http://support.citrix.com/article/CTX118766. Accessed in November 2012.

Duc, G. and R. Keryell, 2005. The concept of secure processes for Linux on the CryptoPage-x86 secure architecture. Technical Report, ENST Bretagne.

Garfinkel, T. and M. Rosenblum, 2003. A virtual machine introspection based architecture for intrusion detection. Proceedings of the Network and Distributed Systems Security Symposium, February 6-7, 2003, San Diego, California, USA., pp: 191-206.

Garfinkel, T., B. Pfaff, J. Chow, M. Rosenblum and D. Boneh, 2003. Terra: A virtual-machine-based platform for trusted computing. Proceedings of the 19th ACM Symposium on Operating Systems Principles, October 19-22, 2003, Bolton Landing, NY, USA., pp: 193-206.

Garfnkel, T., M. Rosenblum and D. Boneh, 2003. Flexible OS support and applications for trusted computing. Proceedings of the 9th USENIX Workshop on Hot Topics on Operating Systems, May 18-21, 2003, Lihue, Hawaii, USA., pp: 145-150.

Goguen, J.A. and J. Meseguer, 1982. Security policies and security models. Proceedings of the IEEE Symposium on Security and Privacy, April 1982, IEEE Computer Society, USA., pp: 11-20.

Jiang, X., X. Wang and D. Xu, 2007. Stealthy malware detection through vmm-based out of-the-box semantic view reconstruction. Proceedings of the 14th ACM Conference on Computer and Communications Security, October 28-31, 2007, Alexandria, Virginia, USA., pp: 128-138.

Jones, S.T., A.C. Arpaci-Dusseau and R.H. Arpaci-Dusseau, 2006. Antfarm: Tracking processes in a virtual machine environment. Proceedings of the Annual Conference on USENIX Annual Technical Conference, May 30-June 3, 2006, Boston, MA, USA.

Krautheim, F J., 2010. Building trust into utility cloud computing. Ph.D. Thesis, University of Maryland, Baltimore, USA.

Loscocco, P.A. and P.W. Wilson, 2007. Linux kernel integrity measurement using contextual inspection. Proceedings of the ACM Workshop on Scalable Trusted Computing, October 29-November 02, 2007, Alexandria, VA., USA., pp: 21-29.

Michael, G., 2010. 10 security concerns for cloud computing. Expert Reference Series of White Papers, Global Knowledge.

Miller, K. and M. Pegah, 2007. Virtualization: Virtually at the desktop. Proceedings of the 35th Annual ACM SIGUCCS Fall Conference, October 7-10, 2007, Orlando, Florida, USA., pp: 255-260.

Payne, B.D., M.D.P. De Carbone and W. Lee, 2007. Secure and flexible monitoring of virtual machines. Proceedings of the 23rd Annual Conference on Computer Security Applications, December 10-14, 2007, Miami Beach, FL., USA., pp: 385-397.

Peinado, M., Y. Chen, P. Engl and J. Manferdelli, 2004. NGSCB: A trusted open system. Proceedings of 9th Australasian Conference on Information Security and Privacy, July 13-15, 2004, Sydney, Australia, pp: 86-97.

Rushby, J., 1992. Noninterference, transitivity and channel-control security policies. SRI International, Computer Science Laboratory. http://csl.sri.com/papers/csl-92-2/csl-92-2.pdf

Sailer, R., X. Zhang, T. Jaeger and L. van Doorn, 2004. Design and implementation of a TCG-based integrity measurement architecture. Proceedings of the 13th USENIX Security Symposium, August 9-13, 2004, San Diego, CA, USA., pp: 223-238.

Santos, N., K.P. Gummadi and R. Rodrigues, 2009. Towards trusted cloud computing. Proceedings of the Conference on Hot Topics in Cloud Computing, June 14-19, 2009, Berkeley, CA, USA.

Scarlata, V., C. Rozas, M. Wiseman, 2008. TPM Virtualization: Building a General Framework. In: Trusted Computing, Pohlmann, N. and H. Reimer (Eds.). Spinger, Germany, pp: 43-56.

TCG, 2003. TCG PC client specific implementation specification. Trusted Computing Group, Version 1.10.

Wojtczuk, R. and J. Rutkowska, 2008. Xen owning trilogy. Proceedings of the Black Hat Conference, July 7, 2008, Las Vegas, NV., USA.

Zhang, X., Y.L. Chen and C.X. Shen, 2009. Non-interference trusted model based on processes. J. Commun., 30: 7-11.

Zhang, X., Q. Huang and C.X. Shen, 2010. A formal method based on noninterference for analyzing trust chain of trusted computing platform. Chin. J. Comput., 33: 74-81.