IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 10, 2017, accepted September 5, 2017, date of publication September 20, 2017,
date of current version October 25, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2754515

Virtualization of the Encryption Card for Trust Access in Cloud Computing

DELIANG XU[1], CAI FU[1], (Member, IEEE), GUOHUI LI[1], DEQING ZOU[1], HONGHAO ZHANG[1], AND XIAO-YANG LIU[2,3]

[1]School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China
[2]Electrical Engineering Department, Columbia University, New York City, NY 10027, USA
[3]Computer Science and Engineering Department, Shanghai Jiao Tong University, Shanghai 200240, China

Corresponding author: Cai Fu (fucai@hust.edu.cn)

This work was supported in part by China NSF under Grant 61572222, Grant 61272405, Grant 61672249, Grant 61772194, Grant 61472121, Grant 61272033, and Grant 61272451, and in part by the Fundamental Research Funds for the Central Universities.

ABSTRACT The increasing use of virtualization puts stringent security requirements on software integrity and workload isolation of cloud computing. The encryption card provides hardware cryptographic services for users and is believed to be superior to software cryptography. However, we cannot use the encryption card directly in the user domain because of the complicated virtualization mechanisms and the security problems about the user key and the user private data flow. To address these challenges, we propose a new virtualization architecture to ensure the trustworthiness of encryption cards. First, we design a privacy preserving model to ensure the security of the dynamic schedule of encryption cards. Second, we present a hardware trust verification procedure based on the trusted platform module to supply a trusted virtualization hardware foundation. Third, we provide a series of security protocols to establish a trusted chain between users and encryption cards. Finally, we give security proofs of the encryption card virtualization architecture. Based on our prototype implementation, the encryption service provided by the encryption card has higher-level security and higher efficiency than software encryption. It provides strong support for security services of virtualization systems in cloud computing.

INDEX TERMS Encryption card, trusted computing, virtualization.

I. INTRODUCTION

Serving as the foundation of cloud computing, in recent years hardware virtualization has gone through a period of rapid development as a method to decrease the total resource cost of owning clouding computing systems [1]. One consequence of the development of virtualization has been the need for rigorous security requirements in the areas of application integrity and user space isolation [2]. The encryption card is a popular hardware device for encrypting and decrypting information. Compared to software encryption, the encryption card offers higher-level security with higher efficiency. The encryption card is physically installed on a host's motherboard and used by the software; it and provides hardware cryptography services for users. People can use advanced algorithms in the encryption card without concerns about the efficiency.

Ideally, the capabilities of an encryption card are available to all users' virtual machines on the same platform [3]. In a perfect world, all virtual machines should be granted virtual access to their own private encryption card, despite the fact that the number of virtual machines usually exceeds the number of physical cards on the system. But the current virtualization system cannot meet the high security requirements in encryption. To defend against possible attacks, the following challenges need to be addressed:

- The security of the encryption resource schedule needs to be protected. Given the limited hardware resources, various users may share one encryption card and one user may use different cards at different points in time. The intricate relationships between users and encryption cards make the resource schedule difficult to execute. We need to design a secure mechanism to manage resource scheduling.
- A malicious user may attack normal users by disguising himself. Because of the lack of authentication, the malicious user can access the normal users' messages by using encryption cards with the normal users' identities. In addition, the malicious user may use up the virtual resources, depriving normal users the use of the encryption services. In these shared hardware environments, malicious users disrupt normal operations, which can

become particularly serious. When different users use one encryption card at the same time, we must be able to distinguish the different users in order to prevent attacks from malicious users.

- The efficiency of an encryption card in virtualization is also an important issue. Current virtualization systems achieve the reuse of hardware devices but they face an efficiency problem. Poor efficiency could degrade the system's performance. We must ensure efficiency high enough to maintain usable virtualization.

Compared with normal devices, encryption card virtualization requires higher-level security, as the users' data is confidential. To manage the encryption card resource schedule, we design a multi-classification access control model to protect users' messages from unauthorized access. To ensure the security of user identification and key transmission from users to encryption cards, we design a series of security protocols to establish a trusted chain between users and encryption cards. We also prove the protocols' correctness through BAN logic [4], a series of rules for formulating and analyzing data exchange protocols. Finally, it is demonstrated through simulation experiments that the virtualization mechanism is efficient enough to avoid a perceptible delay for users.

To summarize, this paper makes the following contributions:

- We establish a risk model to investigate security requirements for encryption card virtualization, including those related to virtual machines and encryption cards.
- We design a multi-classification access control model for security of the encryption resource schedule. In this model, we introduce the concept of virtual classification that achieves secure and flexible access control.
- We design an encryption card virtualization architecture, including secure protocols for user identification and key transmission. The architecture can also be used as a template for virtualization in other security devices.
- We achieve the implementation of the virtualization system, the efficiency of which is comparable to that of the native mode.

The rest of this paper is organized as follows. Section II introduces background concepts that support the subsequent material. Section III, IV, and V respectively describe the design, security analysis, and implementation of the architecture. Section VI concludes the paper.

II. BACKGROUND AND RELATED WORKS

A. BACKGROUND

1) BLP MODEL

The Bell-LaPadula (BLP) Model is a popular state machine model for enforcing access control of applications in government and military. It is a state transition model of computer security policy that includes a set of access control rules that use security levels on clearances for subjects and objects. The Bell-LaPadula model makes no clear distinction between security and protection.

The Bell-LaPadula model pays close attention to data confidentiality and controls access to classified data. In an information system based on the Bell-LaPadula model, entities are divided into objects and subjects. Only if a security policy controls subjects' access to objects, is a system state ''secure.'' To decide if a subject is allowed to access an object, the clearance of a subject is compared to the level of the object. The clearance/level method is expressed in the form of a lattice. The model includes three security properties that define two mandatory access control rules and one discretionary access control rule:

- The Simple Security Property—a subject is not allowed to view an object at a higher security level.
- The *-property—a subject is not allowed to write to an object at a lower security level, nor view an object at a higher security level.
- The Discretionary Security Property—the discretionary access control is specified by an access matrix.

2) HARDWARE VIRTUALIZATION

Virtualization, which dates back to the 1960s, is a method of logically assigning the resources provided by mainframe computers among a various of applications. System partitioning technology created by IBM is a representative example of virtualization [5]. With the development of computer science and the rising demand for computing resources today, advantages of virtualization such as high efficiency are becoming increasingly attractive.

Virtualization conceals the physical components of a computer from the users, providing another virtual computer instead. Through dividing a huge system into small parts, the technology can make a complex physical structure that is difficult to manage, simple and easy to manage. Hardware virtualization is an important part of virtualization, providing architectural support that builds a virtual machine monitor. It also allows users' operating systems to be run in an isolated environment [6]. ''Hypervisor'' is needed to simulate hardware devices for users requesting one device at the same time. The virtual machine transmits users' requests to the hypervisor and the hypervisor handles the requests in order.

3) TRUSTED COMPUTING

Used to build the virtualization mechanism, trusted computing is a method to verify system integrity and ensure information security. With trusted computing, the system behaviors are consistent, following rules enforced by computer hardware and software [7].

As the basis for trusted computing, the Trusted Platform Module (TPM), defined by the Trusted Computing Group [8] as a trusted chain, measures the status of a device and passes the measurements from the BIOS to the boot loader. The measurements are finally passed to the operating system to ensure trust across the system.

The basis of trusted authentication is that a subject gets the proof of an object. Trusted authentication can be described by the formula below:

$$T_r = f_t(\text{proofs, expect}),$$

in which *proofs* are evidence for the subject to verify the object, *expect* is the subject's expectation of what the object is. Function f_t gives a result of whether the object is reliable by comparing the evidence and the expectation.

4) BAN LOGIC

BAN logic is a series of rules that define and analyze information-exchanging protocols [11]. It assists users in distinguishing whether the exchanged information can be trusted and if the information has been eavesdropped. BAN logic is based on the assumption that all information is exchanged via an intermediary that is vulnerable to eavesdropping and tampering. Like all axiomatic systems, BAN logic uses postulates and definitions to analyze authentication protocols. In our project, we adopt BAN logic to analyze a series of security protocols.

When we use BAN logic, we make the following assumptions:

- The ciphertext cannot be falsified or altered; nor can it be formed by a series of small ciphertexts.
- We regard two ciphertexts in a message as ciphertexts that arrive separately
- The encryption system is perfect, i.e., an attacker cannot access the message without knowing the key.
- There is enough redundancy in the ciphertext that the receiver can determine whether the correct key has been used.
- All participants in the protocol are honest.

B. RELATED WORKS

In this section, four research areas related to encryption card virtualization are reviewed: security management based on a virtualization platform, TPM virtualization, TPM-based security architecture, and a method of improving the security of the architecture.

1) SECURITY MANAGEMENT BASED ON VIRTUALIZATION PLATFORM

Krautheim [12] proposed a security model called the Private Virtual Infrastructure for cloud computing that allows the service provider and user to share the responsibility of security, thus reducing the risk exposure to both. However, this model is based on a macroscopic cloud computing security structure, which is not implemented in detail.

Cheng *et al.* [13] presented a protection architecture and integrity measurement, which is used for software stacks that are running on a user operating system of a virtualization platform in the cloud environment. However, due to extensive modifications of hypervisors, which bring a high degree of complexity, this solution does not have good versatility.

Murray *et al.* [14] disaggregated the management virtual machine for a Xen-based system. However, after being disaggregated by a trusted computing base, the system lost some degrees of virtual systems, and may have functional defects because there was no user space.

Sailer *et al.* [15] presented the sHype hypervisor security architecture and examined its mandatory access control facilities. Jansen *et al.* [16] improved the security of virtual machines, specifically in the context of integrity by adding scalable credible computation concepts to a virtual machine infrastructure. However, the two architectures cannot meet the security needs of the encryption card virtualization.

2) TPM VIRTUALIZATION

Berger *et al.* [17] presented the implementation and design of a system that enabled credible computation for virtual machines on a single hardware platform. However, since vTPM security is not assured, the effectiveness of user authentication is greatly compromised.

Kursawe *et al.* [18] proposed a new architecture, that resets the credible boundary to a much lower scale, allowing for much simpler and more flexible uTPM implementations. However, the proposed system is based on a hardware endorsement key to make the signature on the remote authentication, which is likely to expose uTPM hardware information to attackers.

Wan *et al.* [19] proposed an improved secure vTPM migration protocol that uses a credible channel and property-based certification of the destination platform to ensure the security of the vTPM migration. However, this migration solution is implemented in the virtual machines, which cannot offer the same level of security as solutions implemented in the hardware.

Sadeghi *et al.* [20] presented a flexible and privacy-preserving model of a vTPM that in contrast to existing solutions, which supported different methods to measuring the platform's status and for key generation and uses property-based certification mechanisms to support software updates and virtual machine migration.

3) TPM-BASED SECURITY ARCHITECTURE

Fraser and Hand [21] presented a novel secure isolation architecture that uses the latest release of Xen which allowed unmodified hardware device drivers to be shared across isolated instances of operating system, while protecting individual operating systems from the driver failure.

Santos *et al.* [22] proposed a trusted cloud computing platform. He uses TPM to achieve the separation of users from the virtual machine managers. However, the design requires that each virtual machine install a trusted virtual machine monitor module with an embedded TPM chip, leading to high cost.

Azab *et al.* [23] presented a novel framework to enable integrity measurement of a running hypervisor. However, this method cannot be applied to a device virtualization system, because of its disruption of other processes, which may compromise the efficiency of device virtualization systems.

Liu and Deng-Guo [24] presented an integrity measurement architecture that enables administrators to measure the process integrity in system dynamically. Using TPM, the system utilizes a new method to provide dynamical measurement of the processes and kernel modules in system.

Khan *et al.* [25] described the design and implementation of a trusted Eucalyptus cloud architecture that was based on remote attestation via TPM. However, due to the dynamic changes in the data, this solution can only protect the integrity and privacy of user's information temporarily.

4) METHOD OF IMPROVING SECURITY OF TPM-BASED ARCHITECTURE

Stumpf and Eckert [26] presented a credible platform module that supported virtualization based on the hardware techniques. This work introduced a privilege level, only available to a virtual machine monitor, to send control commands to the TPM. However, this approach also makes the trust chain very long, and its reliability is not guaranteed when there is a large number of users.

England and Loeser [27] introduced a technique with a hypervisor through which its guest operating systems can safely share a TPM. However, the trusted chain needs to extend to a virtual machine from a physical TPM application. This also results in a very long trusted chain, in which some components need to be verified repeatedly, thus compromising the effectiveness of the verification process.

Wang and Cheng [28] proposed a technique of access control using a secure virtual machine in Xen architecture. However, this method ignores the new security issues and efficiency problems caused by reusing TPM hardware.

Encryption card virtualization covers all four aspects mentioned above. Since we establish a trusted chain, using TPM as the trusted computing base, TPM virtualization serves as a good reference for our work. When we enhance the security mechanism to address the weakness in traditional virtualization, we consider the security method based on TPM and security management based on a virtualization platform.

III. ENCRYPTION CARD VIRTUALIZATION ARCHITECTURE

We designed a virtual encryption card system that provides encryption card functionality in virtual machines. This section first describes the risk model of virtualization, and then the overall architecture design. It then introduces the virtual Encryption Card Privacy Preserving Model (vEC-PPM) and the TPM-based trust verification mechanism. Finally, it introduces protocols for registration and key transmission.

A. RISK MODEL OF ENCRYPTION CARD VIRTUALIZATION

Before building a secure virtualization architecture, the security threats we expect to encounter must be analyzed. So we will establish a risk model and analyze where the secure condition may be destroyed. After screening, we choose the *Dolev-Yao* model to establish the risk model for encryption card virtualization.

As shown in Fig. 1, the virtualization architecture is represented by a set of abstract domains that can exchange messages. DomU represents users' virtual machines and Dom0 is a manager monitor. EC represents the encryption card. The attacker in this model can overhear, intercept and synthesize any messages and is only constrained by cryptographic methods. Unlike in the real world, the attacker can neither manipulate the bit representation of the encryption nor guess the key. The attacker may re-use any message that has been sent and therefore becomes known. The attacker can encrypt or decrypt these messages with any key he knows and forge subsequent messages.

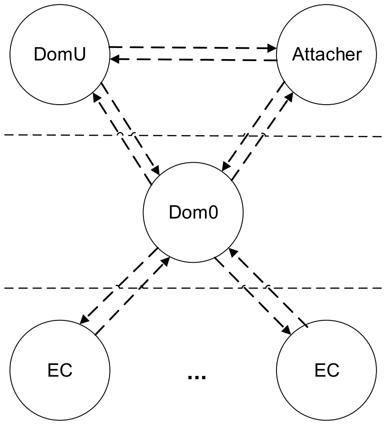


FIGURE 1. Risk model.

We build a risk model in which a user's information passed through certain paths may be under attack. First, the user's key is transmitted to the encryption card through the path $DomU \rightarrow Dom0 \rightarrow EC$. An attacker may intercept the message and get the key through the path $DomU \rightarrow Attacker$. Second, the user's messages are transferred through the path $DomU \rightarrow Dom0 \rightarrow EC$ or $EC \rightarrow Dom0 \rightarrow DomU$. A malicious user may pretend to be a normal user. He communicates with the encryption card through the path $EC \rightarrow Dom0 \rightarrow Attacker$ and then obtain the innocent user's messages. Finally, all users' data is handled in the encryption card. So an attacker may intercept the data through the path $EC \rightarrow Dom0 \rightarrow Attacker$.

B. OVERVIEW OF ENCRYPTION CARD VIRTUALIZATION

As shown in Fig. 2, virtual machines communicate with encryption cards using a split device-driver model, where a frontend driver runs inside each virtual machine that wants to access a virtual encryption card instance.

At the bottom of Fig. 2, the hardware component represents the machine with the TPM and encryption cards. The hypervisor is a virtual machine manager that creates and manages the virtual machines. The backend driver in the hypervisor exchanges messages with the frontend drivers in the virtual machines. The virtual encryption card (vEC) manager in Domain0 allocates virtual encryption card instances for frontend drivers in the virtual machines. We also add a Certification Authority (CA) as a trusted third party that authenticates users and encryption cards. The CA also helps the encryption card to determine whether the user has supplied the correct key.

As illustrated by the dotted arrows in Fig. 2, when a user wants to use an encryption card, the virtual encryption card manager allocates a virtual encryption card instance to the user. This procedure involves both user registration and encryption card registration. Before the user begins to use the

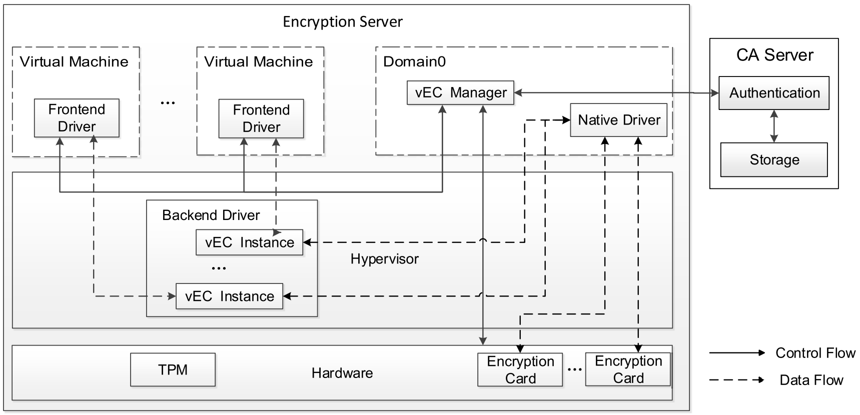


FIGURE 2. Architecture.

encryption card, they verify each other with the help of the CA to establish a trusted relationship. As illustrated by the solid arrows, the frontend drivers transmit users' messages that need to be encrypted to backend driver. The backend driver receives the messages and calls the native driver in Domain0 to use the encryption card.

C. THE vEC PRIVACY PRESERVING MODEL

The vEC Privacy Preserving Model (vEC-PPM) is designed for encryption card virtualization based on the BLP model discussed in Section II. The vEC-PPM redefines elements, security theorems, and state-transition rules. We introduce the dynamic security level policy, which manages the encryption resource schedule. We also introduce the virtual security classification to restrain the adjustment of security level.

1) MODEL ELEMENTS AND VIRTUAL SECURITY CLASSIFICATION

Elements of the vEC-PPM are approximately the same as those of the BLP, including subjects, objects, access attributes, classification, access matrix, and system states. Symbols in the vEC-PPM are the same as those in BLP, unless otherwise indicated.

The components of the vEC-PPM are users, the virtual encryption card manager (vEC-Manager), virtual encryption card instances (vEC-Instance), and encryption cards (EC). These components could be subjects or objects, depending on the direction of the information flow. In particular, The vEC-Manager must be trusted subject (S_T).

We redefine the access attributes as $A = \{r, w, a, e^*, c\}$. The r, w, a attributes are the same as those in BLP. The e^* attribute denotes the vEC-Manager's instruction to an object, such as creating or destroying virtual machines. The c attribute denotes a trust relationship between the subject and the object.

To break the limitation of the traditional static classification, we introduce the virtual security classification, defined below.

Definition 1: The virtual security classification (vL_n) is a discrete-time mapping to static classification, that is $vL_n = \{L_i\}, 1 \leq i \leq p$.

As shown in Fig. 3, a subject or object with virtual security classification can change its current classification depending on the status. A possible situation is an entity's virtual security classification $vL_2 = \{L_3, L_5\}$. When it is handling important data, the current classification is L_3. When it is in free status, the current classification is L_5.

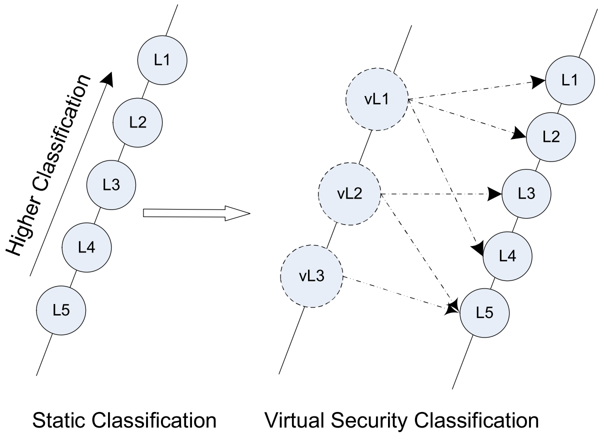


FIGURE 3. Virtual security classification.

We redefine the classification vector as $\{f_s, f_o, f_v\}$, where f_s denotes the clearance of the subject, f_o denotes the classification of the object, and f_v denotes the virtual security classification. The classification of an entity must adjust limited in virtual security classification.

The virtual security classification solves the following problems:

- Applications in one virtual machine have different security levels, so the virtual machine in a static security level cannot serve different applications.
- The BLP is characterized by the phrase "read down, write up." But one encryption card can be distributed to

different users dynamically in virtualization. Encryption cards also need to change their security level.

2) SECURITY THEOREMS

The BLP model includes three security properties that define two mandatory access control rules and one discretionary access control rule: the simple security property, the *-property, and the discretionary security property, respectively. As with the BLP, the vEC-PPM uses three similar security theorems to judge whether a system state is secure. Only if a system state meets all the security conditions in the security theorems, is the state secure. The security theorems are as shown below. Note: All the security theorems in vEC-PPM use "vEC-" as a prefix. Following each security theorem, we provide the proof that every security theorem meets the corresponding theorem in the BLP.

Theorem 1: S' is a subset of S. A state $v = (b \times M \times f \times H)$ meets the vEC-ss-property if

$$s \in S'$$
$$\Rightarrow \{O \in b(S:r,w) \Rightarrow [((S,O,c) \in b)\&(f_s(S) \geq f_o(O))]\}.$$

The vEC-ss-property claims that only if the clearance of a subject is higher than the classification of the object and has the c attribute, can it execute r, w operations.

According to the definition of the vEC-ss-property, $\forall(s,o,x) \in b \wedge [x = r \quad or \quad x = w]$, we get $f_s(s) \geq f_o(o) \wedge (s,o,c) \in b$. Then it is evident that $f_s(s) \geq f_o(o)$. So the vEC-ss-property meets the Simple Security Property in the BLP.

Theorem 2: S' is a subset of S. A state $v = (b \times M \times f \times H)$ meets the vEC--property if*

$$S \in S'$$
$$\Rightarrow \begin{cases} (O \in b(S:a)) \Rightarrow [(f_o(O) \geq f_s(S))\&((S,O,c) \in b)], \\ (O \in b(S:w)) \Rightarrow [(f_o(O) = f_s(S))\&((S,O,c) \in b)], \\ (O \in b(S:r)) \Rightarrow [(f_o(O) \leq f_s(S))\&((S,O,c) \in b)], \\ (O \in b(S:e^*)) \Rightarrow [(f_o(O) \leq f_s(S))\&(S \in S_T)]. \end{cases}$$

To ensure the security of the user data, the vEC-*-property claims that the data flow must be limited to a established trusted chain. Specifically, a subject must have the c attribute to its object if it wants to execute an a, w, r operation.

According to the definition of the vEC-*-property, $\forall(s,o,x) \in b \wedge x = a$, we get $f_s(s) \leq f_o(o) \wedge (s,o,c) \in b$. Then it is evident that $f_s(s) \leq f_o(o)$. Similarly we get $f_s(s) = f_o(o)$ and $f_s(s) \geq f_o(o)$ from $\forall(s,o,x) \in b \wedge x = w$ and $\forall(s,o,x) \in b \wedge x = r$. So the vEC-*-property meets the *-property in BLP.

Theorem 3: A state $v = (b \times M \times f \times H)$ meeting the vEC-ds-property is equivalent to $\forall(s,o,x) \in b$, $x \in M_{ij}$, therein $x \in \{r, a, w, e^*, c\}$.

For the vEC-ds-property, we only change the definition of the e attribute, so the vEC-ds-property meets the ds-property in BLP.

A state must meet all the properties to ensure data security.

3) DYNAMIC SECURITY LEVEL POLICY

In the BLP model, a secure system is based on the *principle of tranquility*. This principle requires that the classification of an entity cannot change once it is created. The principle makes the system secure but not flexible; therefore it cannot work in the dynamic environment in the encryption card virtualization.

In the vEC-PPM, the classification of entities can change under different conditions. Because the applications in one virtual machine have different classifications, the virtual machine has to change its classification in handling different applications. That is why the virtual security classification and dynamic security level policy are needed.

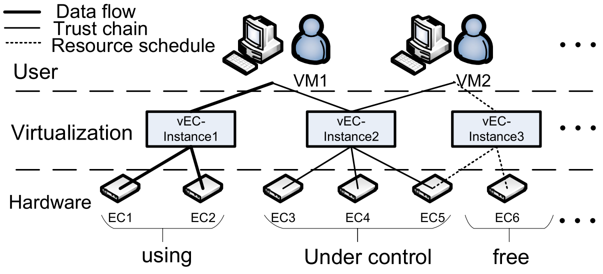


FIGURE 4. **vEC-PPM.**

In Fig. 4, the vEC-PPM is divided into three levels: the hardware level, the middle level, and the user level. Encryption cards are in the hardware level, while vEC-Instances and host operating systems are in the middle level, and virtual machines are in the user level. We define three statuses for the encryption card: free, control, and using.

- free: the encryption card has not mapped with any instance;
- control: the encryption card has mapped with an instance, but it is not handling data;
- using: the encryption card is handling data.

We use the symbol O_f to denote the set of all encryption cards in free status. Similarly, O_c denotes control status and O_u denotes using status.

A user requests a vEC-Instance from the encryption service. Then the vEC-Instance calls the encryption card to handle the user data. A user can use one or more vEC-Instances. A vEC-Instance can also connect with one or more encryption cards. When a user requests more encryption resources, more vEC-Instances or more encryption cards can be providedeither has the same effect.

In particular, if one encryption card is used by several instances, just like EC5 in Fig. 4, it uses a time division multiplexing mechanism, where the encryption card distributes its working time to the instances equally. Of course, before changing to the next instance, the card will clean the context to ensure the information security of users.

Virtual security classification is a special definition in the vEC-PPM. The vEC-Manager has the virtual security classification $vL_1 = \{L_1\}$. It is the only subject at L_1. Encryption cards have the initial virtual security classification

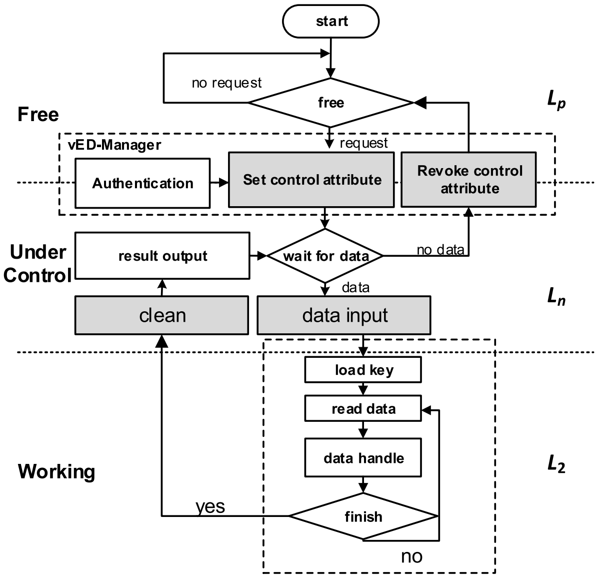


FIGURE 5. State transition.

$vL_2 = \{L_2, L_p\}$. Cards in free status are at L_p, which is the lowest security level. When a card is in the using status, it goes to L_2, which is the highest security level other than vEC-Manager.

Fig. 5 shows the procedure of an encryption card from free to using. At the top of the figure, the encryption card is in the free status, corresponding to L_p. When an instance connects with the encryption card, vEC-Instance gets the c access attribute to the encryption card. At the same time, the card upgrades to the same level as the instance. Now the encryption card is in the control status, and is ready to handle data. When the card receives data from the user (through vEC-Instance), it upgrades to L_2: the specific level for cards to handle data. After finishing handling, the card cleans the user key and other private information and then downgrades to the instance's level to output the results.

4) STATE-TRANSITION RULES

The form of state-transition rules in vEC-PPM is the same as those in the BLP. The input of rules is $(request, state)$ and the output is $(result, state)$. We also call the input as the domain of definition and use the symbol $def(R_i)$ to denote it. Because the *state*s have the same expression $v = (b \times M \times f \times H)$, we only need to focus on the *request*s. In the output, we define *result* as $D = \{yes, no, ?\}$, where *yes* indicates that the *request* is allowed, *no* that it is not, and ? that the *request* is not in the domain of definition.

The rules can be divided into five categories:

- R_{1-5}: The subject gets one of the access attributes to an object.
- R_6: The subject releases a certain access attribute to an object.
- R_{7-8}: The vEC-Manager creates or destroy an object.
- R_{9-10}: The vEC-Manager gives other subjects access attributes.
- R_{11}: The vEC-Manager changes the classification of objects.

R_1 is used to get the c access attribute. The subject (S_i) is the user or the vEC-Instance. The object (O_j) is the vEC-Instance or the encryption card. The subject S_i requests the c access attribute to O_j. The symbol $\leftarrow$ denotes assignment. The symbol $f\backslash f_v$ indicates that f is changed by f_v. If the *result* is *yes*, S_i gets the c access attribute to O_j.

R_{2-5} are used for getting a, w, r, e_* access attributes, respectively. The form of these rules is similar to R_1. We show them together.

R_6 is used for releasing access attributes. The subject (S_i) is the user or the vEC-Instance or the vEC-Manager. The object (O_j) is the vEC-Instance or the encryption card. S_i requests the release of one of the access attributes to O_j. If the *result* is *yes*, S_i releases the access attribute to O_j.

R_7 is used for creating objects. The subject (S_i) is the vEC-Manager. The object (O_j) is the user or the vEC-Instance. If the *result* is *yes*, the vEC-Manager creates an object successfully. (O_j, L_u) denotes $f_o(O_j) = L_u$, while (O_{free}, O_j) indicates that O_j is in the *free* status. After creating the object, the system state upgrades.

R_8 is used for deleting objects. The subject (S_i) is the vEC-Manager. The object (O_j) is the user or the vEC-Instance. If the *result* is *yes*, the vEC-Manager deletes an object successfully. It is worth noting that one of the conditions of *yes* is that the object must be in the *free* status. This condition prevents the loss of user data.

R_9 is used to give access attributes. The subject (S_i) is the vEC-Manager. S_i gives O_j one of the access attributes. O_j can be the user, the vEC-Instance, or the encryption card. If the *result* is *yes*, the corresponding access attribute is added to the access matrix.

R_{10} is used to undo access attributes. The rule is similar to $vEC - R9$. If the *result* is *yes*, the corresponding access attribute is deleted in the access matrix.

R_{11} is used to change the classification of the object. The subject (S_i) is the user or the vEC-Instance or the vEC-Manager. The object (O_j) is the vEC-Instance or the encryption card. The subject (S_i) requests a change in the classification of O_j. $R_*(R_k, v)$ is an indicative function. If state v still meets the vEC-*-property after being changed by R_k, the output of $R_*(R_k, v)$ is *true*. If the *result* is *yes*, S_i changes the classification of the object.

All the state-transition rules are shown in Table I. Only if a transition is allowed by the rules, is the next system state legal.

5) SYSTEM STATE MAP

State-transition rules ensure that the resource redistribution is secure. The system state map is shown in Table 2.

D. TRUST ESTABLISHMENT OF ENCRYPTION CARD VIRTUALIZATION

The trust relationship between users and encryption cards relies on the security provided by TPM [29]. When the encryption server starts up, the TPM verifies measurements

TABLE 1. State-transition rules.

Rule	State	Condition
$R_1(R_k, v)$	$(?, v)$	if$R_k \notin def(R_1)$
	$(yes, (b \cup (S_i, O_j, c), M, f\backslash f_v \leftarrow f_v(O_j) \cup f_s(S_i), H))$	if$[R_k \in def(R_1)]\&[c \in M_{ij}]\&[f_s(S_i) > f_o(O_j)]$
	(no, v)	$otherwise$
$R_2(R_k, v)$	$(?, v)$	if$R_k \notin def(R_2)$
	$(yes, (b \cup (S_i, O_j, a), M, f, H))$	if$[R_k \in def(R_2)]\&[w \in M_{ij}]\&[f_s(S_i) \leq f_o(O_j)]\&[(S_i, O_j, c) \in b]$
	(no, v)	$otherwise$
$R_3(R_k, v)$	$(?, v)$	if$R_k \notin def(R_3)$
	$(yes, (b \cup (S_i, O_j, w), M, f, H))$	if$[R_k \in def(R_3)]\&[w \in M_{ij}]\&[f_s(S_i) = f_o(O_j)]\&[(S_i, O_j, c) \in b]$
	(no, v)	$otherwise$
$R_4(R_k, v)$	$(?, v)$	if$R_k \notin def(R_4)$
	$(yes, (b \cup (S_i, O_j, r), M, f, H))$	if$[R_k \in def(R_4)]\&[r \in M_{ij}]\&[f_s(S_i) \geq f_o(O_j)]\&[(S_i, O_j, c) \in b]$
	(no, v)	$otherwise$
$R_5(R_k, v)$	$(?, v)$	if$R_k \notin def(R_5)$
	$(yes, (b \cup (S_i, O_j, e^*), M, f, H))$	if$[R_k \in def(R_5)]\&[e^* \in M_{ij}]\&[f_s(S_i) \geq f_o(O_j)]\&[S_i \in S_T]$
	(no, v)	$otherwise$
$R_6(R_k, v)$	$(yes, (b - (S_i, O_j, x), M, f, H))$	$[R_k \in def(R_6)]\&x \in \{c, a, w, r, e^*\}$
	$(?, v)$	$otherwise$
$R_7(R_k, v)$	$(?, v)$	if$R_k \notin def(R_7)$
	$(yes, (b, M, f\backslash f_v \leftarrow f_v \cup (O_j, L_u), H \cup (O_j, O_f)))$	if$[R_k \in def(R_7)]\&[S_i \in S_T]$
	(no, v)	$otherwise$
$R_8(R_k, v)$	$(?, v)$	if$R_k \notin def(R_8)$
	$(yes, (b - (S \times \{O_j\} \times A) \cap b - (\{S_j\} \times O \times A) \cap b,$	if$[R_k \in def(R_8)]\&[S_i \in S_T]\&[O_j \notin S_T]\&[O_j \in H(O_u)]$
	$M\backslash\{M_{uj} \leftarrow \phi, M_{ju} \leftarrow \phi\}, f, H - (O_j, O_f)))$	$\&[O_j \notin H(O_c)]$
	(no, v)	$otherwise$
$R_9(R_k, v)$	$(?, v)$	if$R_k \notin def(R_9)$
	$(yes, (b, M\backslash M_{ij} \cup x, f, H))$	if$[R_k \in def(R_9)]\&[S_i = S_T]\&[O_j \in H(O_f)]\&[x \in \{c, a, w, r\}]$
	(no, v)	$otherwise$
$R_{10}(R_k, v)$	$(?, v)$	if$R_k \notin def(R_{10})$
	$(yes, (b - (S_i, O_j, x), M\backslash M_{ij} - x, f, H))$	if$[R_k \in def(R_{10})]\&[x \in \{c, a, w, r\}]\&[S_i = S_T]$
	(no, v)	$otherwise$
$R_{11}(R_k, v)$	$(?, v)$	if$R_k \notin def(R_{11})$
	$(yes, (b, M, f\backslash f_o \leftarrow f_o \cup (O_j, L_u), H \cup (O_r, O_j)))$	if$[R_k \in def(R_{11})]\&[(S_i, O_j, c) \in b]\&[O_j \neq O_r]\&[L_u \in f_v(O_j)]$
		$\&[R_*(R_k, v) = true]$
	(no, v)	$otherwise$

TABLE 2. System state map.

System State	State-Transition Rule
Create VM/vEC-Instance	$vEC - R_7$
Delete VM/vEC-Instance	$vEC - R_8$
User/EC Authentication	$vEC - R_6$
Authentication Success	$vEC - R_9$
Authentication Failed	$vEC - R_{10}$
Data Transmission	$vEC - R_{1-5}$
EC Status Transformation	$vEC - R_{11}$

of the hardware, BIOS, kernel, and operating system in the proper order. If one of the measurements is different from the expected value, the start will be terminated. This process ensures the integrity of the system.

We design three protocols to establish a two-way trust relationship between users and the encryption card. The protocols are designed based on the idea of multiple trust: only if a user believes the encryption card is secure, will the user use it; only if the encryption card verifies that a user is legitimate, will the encryption card serve.

The first two protocols are designed for user registration and encryption card registration. After registration, users' identities and the encryption card's statuses will be stored in the CA.

The third protocol is designed for verification and key loading. The user and the encryption card verify each other with the help of the CA. The user's key will then be transferred to the encryption card securely.

The protocols for user registration and encryption card registration are introduced below, followed by a two-part protocol, including verification (between the user and the encryption card) and key loading.

In this system, there are two kinds of public keys, one for encryption and other for digital signature verification.

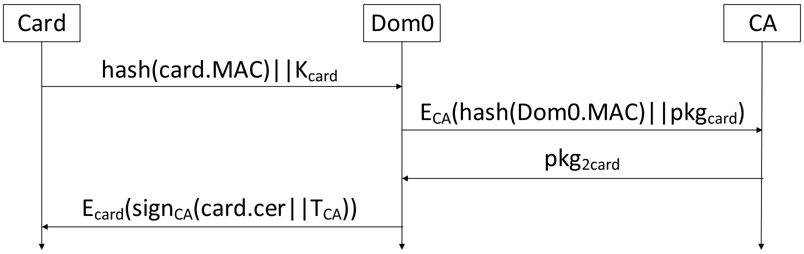

Card
Dom0
CA
hash(card.MAC)||K_{card}
E_{CA}(hash(Dom0.MAC)||pkg_{card})
pkg_{2card}
E_{card}($sign_{CA}$(card.cer||T_{CA}))

FIGURE 6. Card registration.

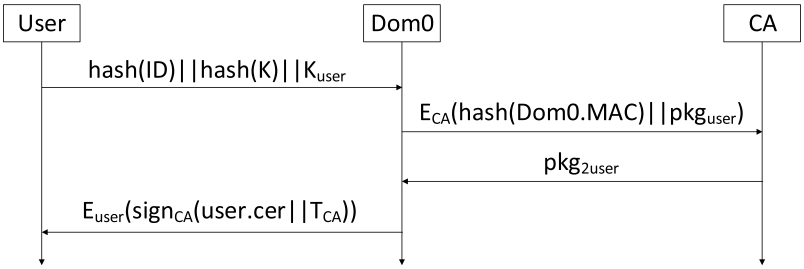

User
Dom0
CA
hash(ID)||hash(K)||K_{user}
E_{CA}(hash(Dom0.MAC)||pkg_{user})
pkg_{2user}
E_{user}($sign_{CA}$(user.cer||T_{CA}))

FIGURE 7. User registration.

1) USER AND ENCRYPTION CARD REGISTRATION

When a new encryption card is installed, it has to register in the CA. After the encryption card sends its status to the CA, the CA logs the card's information and issues a certificate to the card. Now the encryption card is certified and ready to be used. As shown in Fig. 6, the encryption card registration protocol includes the following procedures:

1) The encryption card sends the hash of its identity marker to Dom0. In most cases, we choose the MAC of the encryption card as its identity marker, as MAC is unique for each encryption card. The public key of the encryption card, K_{card}, is also sent to Dom0.
2) Dom0 adds its own identity marker (also using MAC) to the package and encrypts the whole new package using the public key of the CA. Dom0 then sends the encrypted package to the CA.
3) After receiving and decrypting the package, the CA logs the status of encryption card and generates a certificate for the card with a time stamp and a signature. Finally, the CA encrypts the whole package using the public key of the encryption card and sends the encrypted package to Dom0.
4) Dom0 passes the package to the encryption card. The card decrypts the package and obtains its certificate.

As shown in Fig. 7, the procedures of user registration are similar to those of the encryption card, except that the user also sends the hash of his or her encryption key to the CA.

2) VERIFICATION AND KEY LOADING

This protocol consists of two parts. Only if part 1 is completed successfully can part 2 be started. Part 1 checks whether the user and its corresponding encryption card are registered; part 2 passes the user's encryption key to his or her encryption card. The procedures of the protocol are shown in Fig. 8 and are detailed as follows:

Part 1: Verification

1) The user and the corresponding encryption card sign on the hash of their respective identity markers and send them to Dom0.
2) Dom0 encrypts the two packages from the user and the encryption card using the public key of the CA and then sends the whole encrypted package to the CA.
3) After receiving and decrypting the package, the CA determines whether the user and the corresponding card are registered by checking their identities against information in storage. The two results of this checking process for the user and the corresponding card are timestamped and encrypted using the public keys of the user and the encryption card, respectively. Finally, the two encrypted packages are sent to Dom0.
4) Dom0 receives the two packages and passes them to the user and the corresponding encryption card, respectively.
5) The user and the card receive the results packages and decrypt them. If both the user and the encryption card have been registered, part 2 of the protocol will be started.

Part 2: Key Loading

1) The user encrypts his or her encryption key using the public key of the encryption card and then sends the encrypted package to the encryption card.
2) The encryption card decrypts the package and gets the user's encryption key. The card then sends the hash of the user's encryption key to the CA to determine if the key is correct for authentication.

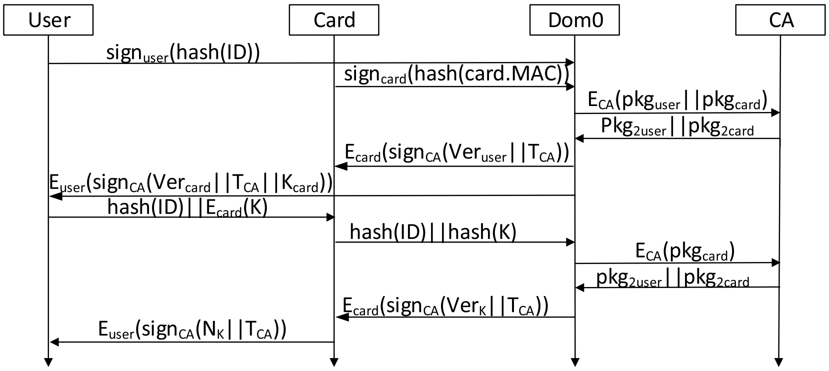

User
Card
Dom0
CA
$\text{sign}_{\text{user}}(\text{hash(ID)})$
$\text{sign}_{\text{card}}(\text{hash(card.MAC)})$
$E_{CA}(\text{pkg}_{\text{user}}||\text{pkg}_{\text{card}})$
$\text{Pkg}_{\text{2user}}||\text{pkg}_{\text{2card}}$
$E_{\text{card}}(\text{sign}_{CA}(\text{Ver}_{\text{user}}||T_{CA}))$
$E_{\text{user}}(\text{sign}_{CA}(\text{Ver}_{\text{card}}||T_{CA}||K_{\text{card}}))$
$\text{hash(ID)}||E_{\text{card}}(K)$
$\text{hash(ID)}||\text{hash}(K)$
$E_{CA}(\text{pkg}_{\text{card}})$
$\text{pkg}_{\text{2user}}||\text{pkg}_{\text{2card}}$
$E_{\text{card}}(\text{sign}_{CA}(\text{Ver}_K||T_{CA}))$
$E_{\text{user}}(\text{sign}_{CA}(N_K||T_{CA}))$

FIGURE 8. Verification and key loading.

3) The CA receives the card's request and decrypts it. The CA then compares the received hash with the hash in storage. Finally, the CA signs and encrypts the checking results which are sent to Dom0. An encryption key number is also encrypted and sent to Dom0.
4) The user and the card each receive their results. The encryption key number will be used as an identifier in communication between the user and the corresponding card.

IV. MODEL ANALYSIS

This section proves the integrity of the design. It contains the analysis of vEC-PPM, the analysis of encryption cards and users registration, and the analysis of verification and key loading. By demonstrating, we reach the conclusion that the state-transition rules in vEC-PPM and the protocols in trust establishment are secure.

A. vEC-PPM SECURITY ANALYSIS

The vEC-PPM is a finite-state machine model based on BLP. Only if a state in the vEC-PPM meets the three security theorems, is the state secure. According to the vEC-ss-property, a subject is not allowed to view an object at a security level higher than his or her own. According to the vEC-*-property, a subject is not allowed to write to an object at a lower security level, nor view an object at a higher security level. According to the vEC-ds-property, the operation that a subject to an object has to meet the access matrix. We give the security proof for $vEC - R_4$ and $vEC - R_{11}$ as examples to prove that the state-transition rules are secure.

We assume that $vEC - R_4(R_k, v) = (D_m, v')$, therein v is a secure state that meets vEC-*-property, vEC-ss-property, and vEC-ds-property. We get $v' = v$ or $v' = (b \cup \{(S_i, O_j, r)\}, M, f, H)$.

1) $v' = v$
It is evident that v' meets the three properties.
2) $v' = (b \cup \{(S_i, O_j, r)\}, M, f, H)$
If $(S_i, O_j, r) \in b$, then $v' = v$ and v' meets the three properties. If $(S_i, O_j, r) \notin b$, according to $vEC - R_4$ we get $f_s(S_i) >= F_o(O_j)$ and $(S_i, O_j, c) \in b$. So the rule meets the vEC-*-property. For $\forall O \in b'(S : r)$, because $b' = b \cup (S_i, O_j, r)$, that is $(S_i, O_j, r) \in b'$, we get $((S, O, c) \in b')\&(f_s(S) >= f_o(O))$. So the rule meets vEC-ss-property. Last, we get $r \in M_{ij}$ from the rule. The rule meets the vEC-ds-property.

In conclusion, the $vEC - R_4$ meets the three security properties and the Basic Security Theorem.

We assume that $vEC - R_{11}(R_k, v) = (D_m, v')$, therein v is a secure state that meets vEC-*-property, vEC-ss-property, and vEC-ds-property. We get $v' = v$ or $v' = (b, M, f \backslash f_o \leftarrow f_o \cup (O_j, L_u), H \cup (O_r, O_j))$.

1) $v' = v$
It is evident that v' meets the three properties.
2) $v' = (b, M, f \backslash f_o \leftarrow f_o \cup (O_j, L_u), H \cup (O_r, O_j))$
Because the rule does not add a new element to b, v' inherits the vEC-ds-property from v. According to $vEC - *11(R_k, v) = true$, for $\forall S \in b(O_j : w, a)$, we get $L_u \geq f_s(S)$. For $\forall S \in b(O_j : w, r)$, we get $L_u \leq f_s(S)$. For $\forall O \in b(S_j : w, a)$, we get $L_u \leq f_o(O)$. So the rule meets the vEC-*-property and vEC-ss-property.

In conclusion, the $vEC - R_{11}$ meets the three security properties and the Basic Security Theorem.

Similarly, we can give proof that the other rules are secure.

B. ENCRYPTION CARDS AND USERS REGISTRATION ANALYSIS

In this section, we give security proof of encryption cards and user registration protocols by using BAN logic. By derivation, the two protocols are both secure.

1) IDEALIZATION OF ENCRYPTION CARD REGISTRATION

The user registration protocol is shown below, where the messages are derived from the card registration process described in section 3.4.1 (see Fig. 6 for details).

$$\begin{aligned} &\textit{Message 1} \quad \textit{Card} \rightarrow \textit{Dom0}: \\ &\qquad \textit{Card}, K_{card} \\ &\textit{Message 2} \quad \textit{Dom0} \rightarrow \textit{CA}: \\ &\qquad \{\textit{Dom0}, \textit{Card}, K_{card}\}_{K_{CA}} \end{aligned}$$

$$\begin{aligned}&Message\ 3 \quad CA \rightarrow Dom0:\\ &\quad \{\{card.cer, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}}\\ &Message\ 4 \quad Dom0 \rightarrow Card:\\ &\quad \{\{card.cer, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}}.\end{aligned}$$

It is assumed that Dom0 is reliable, i.e., it only passes the cards' messages to the CA or encrypted information back to the card. Therefore, we omit Dom0 in the idealized form and, in our analysis, the card communicates with the CA directly. We idealize the protocol as follows:

$$\begin{aligned}&Message\ 2 \quad Card \rightarrow CA:\\ &\quad \{Card, \xrightarrow{K_{card}} card\}_{K_{CA}}\\ &Message\ 4 \quad CA \rightarrow Card:\\ &\quad \{\{card.cer, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}}.\end{aligned}$$

2) PROTOCOL ANALYSIS OF ENCRYPTION CARD REGISTRATION

First, the following assumptions on trust are made. We use A | ≡ B to indicate that A trusts B and # M to indicate that the message M is fresh.

$$\begin{aligned}card| &\equiv K_{CA}\\ CA| &\equiv K_{card}\\ CA| &\equiv card.cer\\ card| &\equiv (CA| \Rightarrow card.cer)\\ card| &\equiv \#(T_{CA}).\end{aligned}$$

The encryption card sends its identity to the CA, which authenticates and certifies it. The CA passes the certification to the card, along with a time stamp T_{CA}. The card receives the encrypted message and decrypts it, i.e.,

$$card \lhd \{card.cer, T_{CA}\}_{K_{CA}^{-1}},$$

where A ◃ B indicates that A has received B.

Knowledge of the CA's public key allows the card to decrypt the message above:

$$card \lhd (card.cer, T_{CA}).$$

By applying the rules of *message-meaning*, *nonce-verification*, and jurisdiction [11], the card trusts the certificate from the CA, i.e.,

$$card| \equiv card.cer.$$

This concludes the analysis of card registration. The analysis of user registration is omitted in the interest of space, as it is similar to that of cards registration.

C. VERIFICATION AND KEY LOADING ANALYSIS

In this section, we give security proof of verification and key loading protocol by using BAN logic. By derivation, the protocol is secure.

1) IDEALIZATION OF VERIFICATION AND KEY LOADING

This protocol is shown below. Messages are derived from Fig. 8. As this protocol consists of two parts, we will use the result of part 1 when we analyze part 2.

Part 1: Part 1 is as follows:

$$\begin{aligned}&Message\ 1 \quad VM \rightarrow Dom0:\\ &\quad \{User\}_{K_{user}^{-1}}\\ &Message\ 2 \quad Card \rightarrow Dom0:\\ &\quad \{Card\}_{K_{card}^{-1}}\\ &Message\ 3 \quad Dom0 \rightarrow CA:\\ &\quad (\{User\}_{K_{user}^{-1}}, \{Card\}_{K_{card}^{-1}})_{K_{CA}}\\ &Message\ 4 \quad CA \rightarrow Dom0:\\ &\quad (\{\{Ver_{user}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}},\\ &\quad \{Ver_{card}, K_{card}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}})\\ &Message\ 5 \quad Dom0 \rightarrow Card:\\ &\quad \{\{Ver_{user}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}}\\ &Message\ 6 \quad Dom0 \rightarrow VM:\\ &\quad \{\{Ver_{card}, K_{card}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{user}}.\end{aligned}$$

Part 2: Part 2 is as follows:

$$\begin{aligned}&Message\ 7 \quad VM \rightarrow Card:\\ &\quad (User, \{K\}_{K_{card}})\\ &Message\ 8 \quad Card \rightarrow Dom0:\\ &\quad (User, hash(K))\\ &Message\ 9 \quad Dom0 \rightarrow CA:\\ &\quad (User, hash(K))_{K_{CA}}\\ &Message\ 10 \quad CA \rightarrow Dom0:\\ &\quad (\{\{Ver_{K}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}},\\ &\quad \{\{N_{K}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{user}})\\ &Message\ 11 \quad Dom0 \rightarrow Card:\\ &\quad \{\{Ver_{K}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}}\\ &Message\ 12 \quad Dom0 \rightarrow VM:\\ &\quad \{\{N_{K}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{user}}.\end{aligned}$$

Similarly to 4.2.1, we also omit Dom0 in the idealized form and have the card communicates with the CA directly. We idealize messages in two parts in the protocol. In the analysis, we will use the result of part 1 when we analyze part 2.

Part 1: Part 1 is as follows:

$$\begin{aligned}&Message\ 1 \quad VM \rightarrow CA:\\ &\quad \{User\}_{K_{user}^{-1}}\\ &Message\ 2 \quad Card \rightarrow CA:\\ &\quad \{Card\}_{K_{card}^{-1}}\end{aligned}$$

$$\begin{aligned}
&Message\ 5 \quad CA \rightarrow Card: \\
&\quad \{\{Ver_{user}, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}} \\
&Message\ 6 \quad CA \rightarrow VM: \\
&\quad \{\{Ver_{card}, \xrightarrow{K_{card}} card, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{user}}.
\end{aligned}$$

Part 2: Part 2 is as follows:

$$\begin{aligned}
&Message\ 7 \quad VM \rightarrow Card: \\
&\quad (User, \{K\}_{K_{card}}) \\
&Message\ 8 \quad Card \rightarrow CA: \\
&\quad (User, hash(K)) \\
&Message\ 11 \quad CA \rightarrow Card: \\
&\quad \{\{Ver_K, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{card}} \\
&Message\ 12 \quad CA \rightarrow VM: \\
&\quad \{\{N_K, T_{CA}\}_{K_{CA}^{-1}}\}_{K_{user}}.
\end{aligned}$$

2) PROTOCOL ANALYSIS OF VERIFICATION AND KEY LOADING

Part 1: The following assumptions are made about the verification protocol.

$$\begin{aligned}
user| &\equiv \xrightarrow{K_{CA}} CA \\
card| &\equiv \xrightarrow{K_{CA}} CA \\
CA| &\equiv \xrightarrow{K_{user}} user \\
CA| &\equiv \xrightarrow{K_{card}} card \\
user| &\equiv (CA| \Rightarrow Ver_{card}) \\
card| &\equiv (CA| \Rightarrow Ver_{user}) \\
user| &\equiv (CA| \Rightarrow (\xrightarrow{K_{card}} card)) \\
user| &\equiv \#(T_{CA}) \\
card| &\equiv \#(T_{CA}).
\end{aligned}$$

The user and the encryption card send their identities to the CA, which checks and makes sure that the card has registered, then sends a verification to the user along with the public key of the card, which will be used in Part 2. The user receives message 6, and from the rule of *message-meaning*, *nonce-verification*, and *jurisdiction*, we obtain:

$$\begin{aligned}
user| &\equiv Ver_{card}, \\
user| &\equiv K_{card},
\end{aligned}$$

which means the user believes the card is legitimate.

Similarly, the card receives a verification assuring it that the user is reliable:

$$card| \equiv Ver_{user}.$$

Part 2: The following assumptions are made about the key loading protocol.

$$\begin{aligned}
user| &\equiv \xrightarrow{K_{CA}} CA \\
card| &\equiv \xrightarrow{K_{CA}} CA \\
CA| &\equiv \xrightarrow{K_{user}} user \\
CA| &\equiv \xrightarrow{K_{card}} card \\
user| &\equiv (CA| \Rightarrow N_K) \\
card| &\equiv (CA| \Rightarrow Ver_K) \\
user| &\equiv \#(T_{CA}) \\
card| &\equiv \#(T_{CA}).
\end{aligned}$$

When the card receives K, it passes the hash of K to the CA for authentication. The CA then sends a verification to the card to confirm that K is correct. In the same way, the final result is:

$$card| \equiv Ver_K.$$

The user also gets the identifier of K:

$$user| \equiv N_K.$$

This concludes the analysis of the processes of verification and key loading.

V. IMPLEMENTATION

After actualizing the virtualization of the encryption card in the KVM platform, we add the vEC-PPM to key routes in the system as an extra security model. The implementation of the vEC-PPM is shown in Fig. 9.

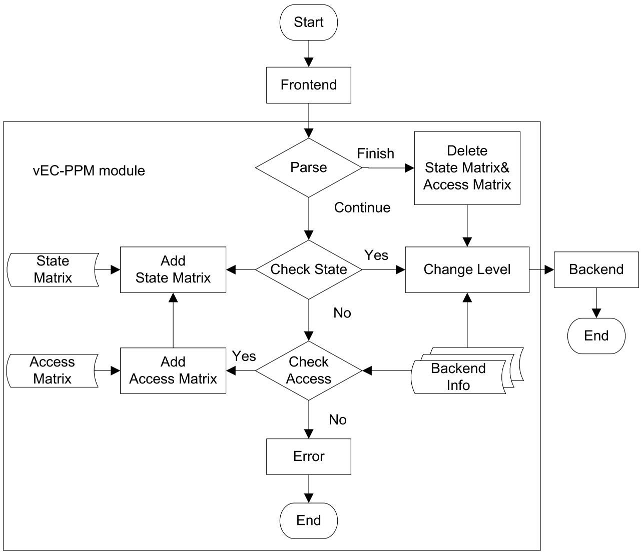
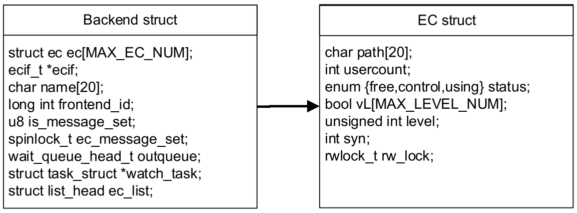


FIGURE 9. Implementation of vEC-PPM.

The vEC-PPM is initialized after the backend driver is loaded. The model creates two arrays to denote the access matrix and the state matrix. We use five bits in one byte to denote access attributes r, a, w, e^*, and c. To save the mapping between virtual machines and encryption cards, we change codes in KVM. The key data structure is shown in Fig. 10.

The EC structure saves the key information of an encryption card. It is worth noting that the virtual security classification is saved in a bool array. In our implementation, we



```
struct ec ec[MAX_EC_NUM];
ecif_t *ecif;
char name[20];
long int frontend_id;
u8 is_message_set;
spinlock_t ec_message_set;
wait_queue_head_t outqueue;
struct task_struct *watch_task;
struct list_head ec_list;
```

```
char path[20];
int usercount;
enum {free,control,using} status;
bool vL[MAX_LEVEL_NUM];
unsigned int level;
int syn;
rwlock_t rw_lock;
```

FIGURE 10. Data structure.

TABLE 3. Server configuration.

Component	Configuration
CPU	Intel Core i3-4160 @ 3.60 GHz
Memory Capacity	4GB
Disk Capacity	500GB
Operating System	openSUSE 12.3
System Kernel	Linux 3.7.10
Hypervisor	Qemu-KVM 1.2.0
Encryption Card	SWCSM09

TABLE 4. Virtual machine configuration.

Component	Configuration
CPU	Intel Core i3-4160 @ 3.60 GHz
Virtual CPU Quantity	1
Memory Capacity	2GB
Disk Capacity	20GB
Operating System	Windows XP Professional SP3

set $L = \{L_1, L_2, L_3, L_4\}$. L_1 is the highest level, used only for vEC-Manager, L_2 is for encryption cards in using status, L_3 is for virtual machines, and L_4 is for cards in free status.

Experiments are conducted in two parts: *initialization time* and *efficiency*. Because of the vEC-PPM and related security mechanism, initialization time in a virtual machine is inevitably longer than that of the native model. Efficiency is also the key concern that is reflected in the testing. The configuration of the server and virtual machines is detailed in Table 3 and Table 4, respectively. It is worth noting that the source of the encryption card is Sanwei Xin'an Company. The card provides RSA, AES, and SM1. We use the SM1 algorithm in our implementation.

A. INITIALIZATION

According to our design, before users' keys can be transmitted to the encryption card, users and the encryption card need to be authenticated. After the virtual machine starts up, a series of authentication processes are required to transmit the keys in a safe environment. Inevitably, these processes affect the efficiency of the encryption card. The purpose of this experiment is to quantify this delay.

The experiments are repeated ten times under the same environment and in the same virtual machine. By repeatedly switching on the virtual machine, we can add up the time taken for each authentication to calculate the total initialization time of the virtual machine.

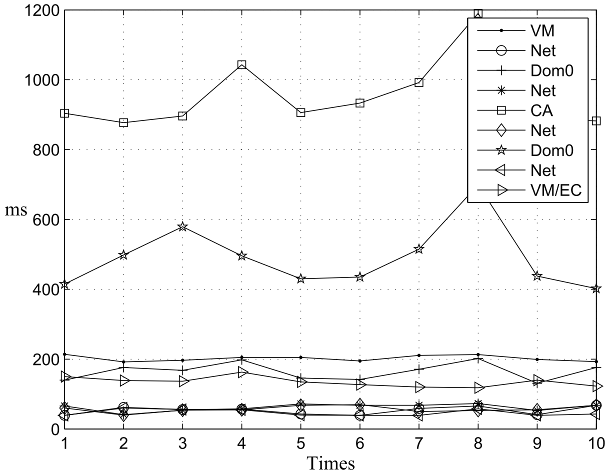


FIGURE 11. Verification time.

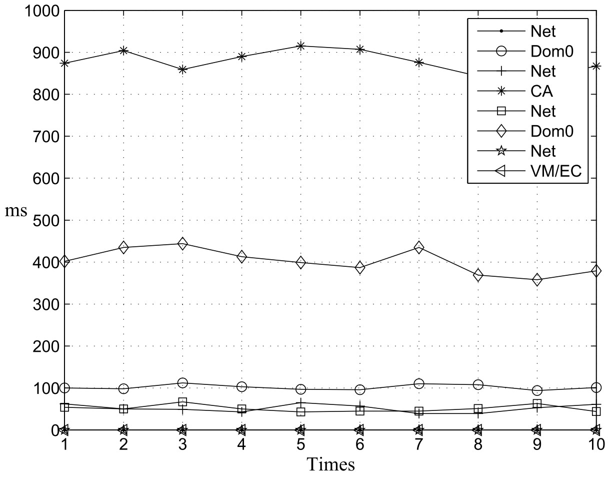


FIGURE 12. Loading time.

Results of the experiments are presented in Fig. 11 and Fig. 12, which show the initialization time of each authentication stage. It can be seen that the average time of each stage is generally acceptable. For example, the network transmission time, which refers to each request's transfer time between the virtual machine, Dom0, and the CA, is around 50 ms and the minimum time is 0. In particular, if the time equals 0, this indicates that information transmission in this stage is performed by the frontend driver, so the time is negligible. Refer to the data indicate that there is little variation in initialization time consumption, the average of which is about 3,648 ms, exerting little influence on the practical user experience.

Fig. 13 lists the average time of each stage in the experiments, after being repeated ten times. Obviously, the process times of the CA and Dom0 are longer than others. For example, the times of the CA are up to 951 ms and 877 ms, which together take up 50 percent of the total initialization time causing a bottleneck. During initialization, the CA authenticates users' contextual information and encryption

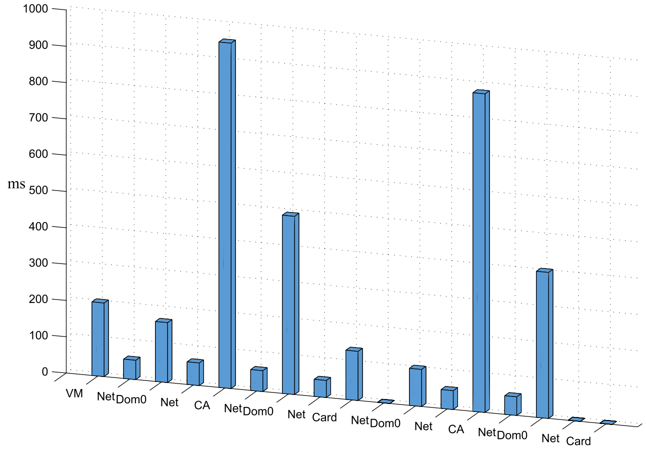


FIGURE 13. Initializing time.

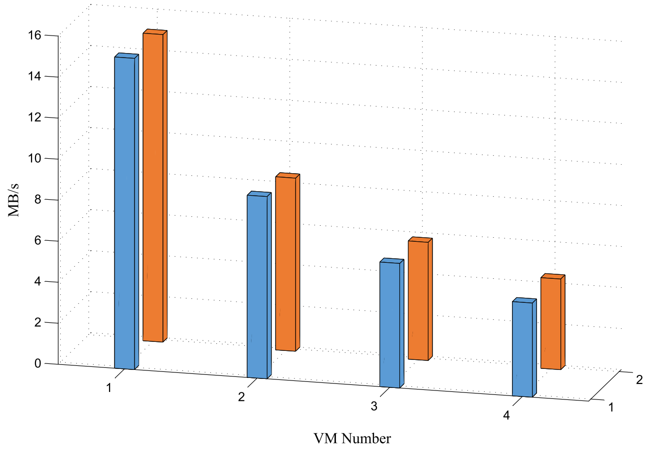


FIGURE 14. Average speed in virtualization.

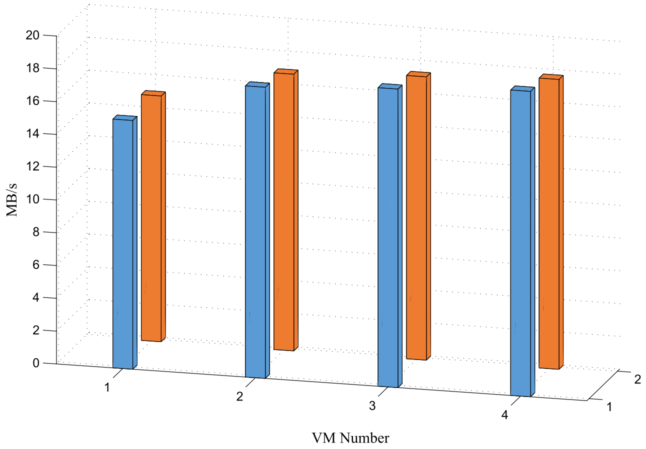


FIGURE 15. Total speed in virtualization.

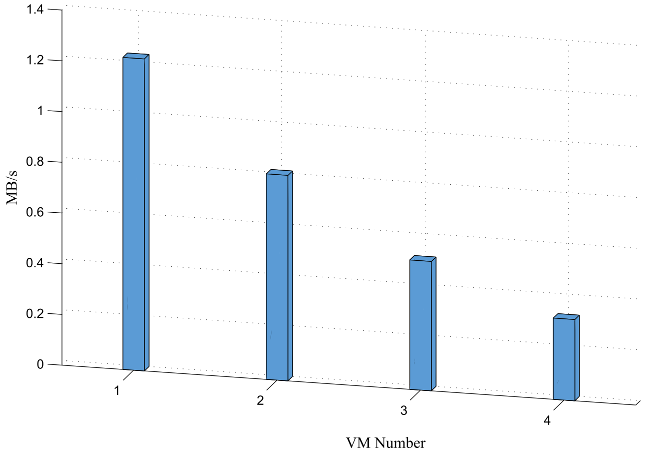


FIGURE 16. Average speed in non-virtualization.

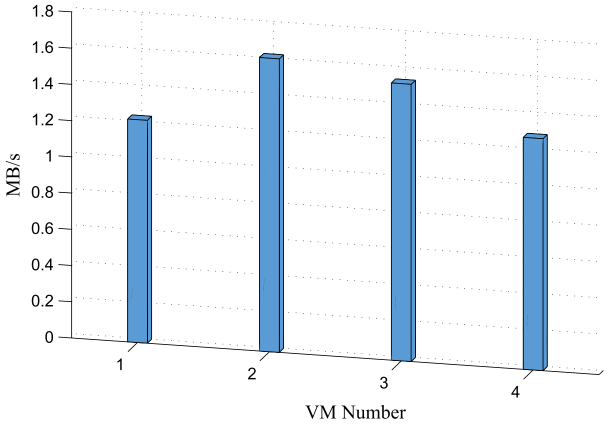


FIGURE 17. Total speed in non-virtualization.

card respectively, and then authenticates users' keys, each of which involve multiple instances of decryption and encryption. If the efficiency of the CA could be improved, initialization time could be significantly reduced.

B. PERFORMANCE

Given the great impact of virtualization on the efficiency of the overall system, these experiments are conducted to quantify the decrease of efficiency when additional virtual machines are introduced into the system.

In virtualization, one encryption card is shared across multiple virtual machines. Fig. 14 shows how the average encrypting speed of each virtual machine changes when the number of virtual machine increases. It shows that when there is only one virtual machine, the average encrypting speed is 15.17 MB/s without vEC-PPM and 14.99 MB/s with vEC-PPM. When more virtual machines share the encryption card, each virtual machine's average encrypting speed decreases to 4.59 MB/s and 4.59 MB/s, respectively.

Fig. 15 shows the total encrypting speed of virtual machines when the number of virtual machine increases. When the number of virtual machines increases to four, the total speed increases to 18.36 MB/s. In general, the total encrypting speed remains steady throughout the experiments suggesting that there is very little loss of efficiency, even in a multiple virtual machines mode.

In contrast, we measure the speed of software encryption. Fig. 16 and Fig. 17 show the virtual machines' average speeds and total speeds respectively. The trend is similar to that of hardware encryption. We assume that the software and

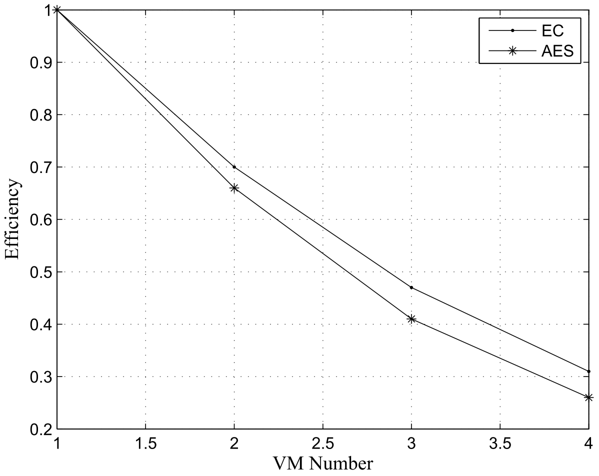


FIGURE 18. Speed comparison between software and hardware encryption.

hardware encryption speed are both one unit when there is one virtual machine. The trends in the decrease in speed for the two encryption modes are shown in Fig. 18, which indicates that the encrypting speed decreases more slowly in hardware mode than in software mode.

When we use software encryption, the CPU schedules the computing resources and does the encryption at the same time. Heavy scheduling tasks will grab the CPU computing resources. But when we use hardware encryption, the CPU only schedules the computing resources: the encryption is done by encryption cards and the efficiency of encryption will not be affected by scheduling. Hence there is higher efficiency if we use encryption card.

In conclusion, through encryption card virtualization, our encryption card can support multiple virtual machines efficiently, and there is very little loss of efficiency due to the vEC-PPM.

VI. CONCLUSION

Hardware virtualization is an important technology for cloud computing, where security is currently provided through software encryption. However, this technology is confronted with many risks: for example, the user's encryption key may be disclosed, and the user's private data may be exposed to malicious users. Higher-level security is needed, especially when important information is encrypted. Compared to software encryption, an encryption card has higher-level security and higher efficiency. Nevertheless, with many problems to be solved, encryption card virtualization has not been extensively implemented despite its technological advantages.

In this paper, we designed a virtual encryption card system that provides encryption card functionality in virtual machines. In this system, we presented the vEC-PPM, which manages the encryption resource schedule. We preserved users' data using a trusted hardware of virtualization based on TPM. We also established a trusted chain between users and encryption cards based on the designed protocols. Our design of the virtual encryption card enables the security and efficiency of the encryption service. An implementation analysis demonstrates that the efficiency of our system is comparable to that of the native mode.

In the future, we will continue with our study, seeking to design a virtual encryption cards cluster to support higher encryption velocity and more suitable compatibility with virtualization.

REFERENCES

[1] R. Figueiredo, P. A. Dinda, and J. Fortes, "Resource virtualization renaissance," *IEEE Comput.*, vol. 38, no. 5, pp. 28–31, May 2005.

[2] D. Sgandurra and E. Lupu, "Evolution of attacks, threat models, and solutions for virtualized systems," *ACM Comput. Surv.*, vol. 48, no. 3, 2016, Art. no. 46.

[3] S.-K. Kim, S.-Y. Ma, and J. Moon, "A novel secure architecture of the virtualized server system," *J. Supercomput.*, vol. 72, no. 1, pp. 24–37, 2015.

[4] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," *Proc. Roy. Soc. A Math. Phys. Eng. Sci.*, vol. 426, no. 1871, pp. 233–271, 1989.

[5] R. Creasy, "The origin of the VM/370 time-sharing system," *IBM J. Res. Develop.*, vol. 25, no. 5, pp. 483–490, Sep. 1981.

[6] R. Uhlig *et al.*, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, May 2005.

[7] C. Mitchell, *Trusted Computing*. London, U.K.: IEE, 2005.

[8] Trusted Computing Group. *Trusted Platform Module Library Part 1: Architecture*. Accessed: Sep. 29, 2016. [Online] Available: https://trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-1-Architecture-01.38.pdf

[9] J. A. Goguen and J. Meseguer, "Security policies and security models," in *Proc. Symp. Secur. Privacy*, Oakland, CA, USA, Apr. 1982, p. 11.

[10] J. A. Goguen and J. Meseguer, "Unwinding and inference control," in *Proc. Symp. Secur. Privacy*, Oakland, CA, USA, Apr. 1984, p. 75.

[11] M. Burrows, M. Abadi, and R. Needham, "A logic of authentication," in *Proc. 12th ACM Symp. Oper. Syst. Principles (SOSP)*, New York, NY, USA, 1989, pp. 1–13.

[12] F. J. Krautheim, "Private virtual infrastructure for cloud computing," in *Proc. Conf. Hot Topics Cloud Comput. USENIX Assoc.*, 2009, pp. 1–5.

[13] G. Cheng, H. Jin, D. Zou, and X. Zhang, "Building dynamic and transparent integrity measurement and protection for virtualized platform in cloud computing," *Concurrency Comput., Pract. Exper.*, vol. 22, no. 9, pp. 1893–1910, 2010.

[14] D. G. Murray, G. Milos, and S. Hand, "Improving Xen security through disaggregation," in *Proc. VEE*, New York, NY, USA, 2008, pp. 151–160.

[15] R. Sailer *et al.*, "Building a MAC-based security architecture for the Xen open-source hypervisor," in *Proc. ACSAC*, Washington, DC, USA, Dec. 2005, p. 285.

[16] B. Jansen, H. V. Ramasamy, and M. Schunter, "Flexible integrity protection and verification architecture for virtual machine monitors," in *Proc. 2nd Workshop Adv. Trusted Comput.*, 2006, pp. 1–13.

[17] S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, "vTPM: Virtualizing the trusted platform module," in *Proc. 15th Conf. USENIX Secur. Symp.*, 2006, pp. 305–320.

[18] K. Kursawe and D. Schellekens, "Flexible μTPMs through disembedding," in *Proc. ACM Symp. Inf., Comput. Commun. Secur.*, 2009, pp. 116–124.

[19] X. Wan, X. Zhang, L. Chen, and J. Zhu, "An improved vTPM migration protocol based trusted channel," in *Proc. Int. Conf. Syst. Inform. (ICSAI)*, May 2012, pp. 870–875.

[20] A. R. Sadeghi, C. Stüble, and M. Winandy, "Property-based TPM virtualization," in *Proc. 11th Int. Conf. Inf. Secur. (ISC)*, 2008, pp. 1–16.

[21] K. Fraser, S. Hand, R. Neugebauer, I. Pratt, A. Wareld, and M. Williamson, "Safe hardware access with the Xen virtual machine monitor," in *Proc. 1st Workshop Oper. Syst. Archit. Support Demand IT InfraStruct. (OASIS)*, Boston, MA, USA, Oct. 2004, pp. 1–10

[22] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proc. Conf. Hot Topics Cloud Comput. USENIX Assoc.*, Berkeley, CA, USA, 2009, pp. 1–5.

[23] A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky, "HyperSentry: Enabling stealthy in-context measurement of hypervisor integrity," in *Proc. 17th ACM Conf. Comput. Commun. Secur.*, 2010, pp. 38–49.

[24] Z.-W. Liu and D.-G. Feng, "TPM-based dynamic integrity measurement architecture," *J. Electron. Inf. Technol.*, vol. 32, no. 4, pp. 875–879, 2010.

[25] I. Khan, H. Rehman, and Z. Anwar, "Design and deployment of a trusted eucalyptus cloud," in *Proc. IEEE Int. Conf. Cloud Comput.*, Washington, DC, USA, Jul. 2011, pp. 380–387.

[26] F. Stumpf and C. Eckert, "Enhancing trusted platform modules with hardware-based virtualization techniques," in *Proc. 2nd Int. Conf. Emerg. Secur. Inf., Syst. Technol.*, Aug. 2008, pp. 1–9.

[27] P. England and J. Loeser, "Para-virtualized TPM sharing," in *Proc. 1st Int. Conf. Trusted Comput. Trust Inf. Technol.*, 2008, pp. 119–132.

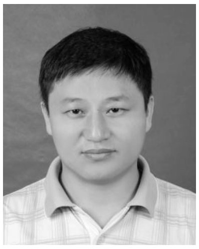
[28] X. Wang and C. Cheng, "Access control using trusted virtual machine based on xen," in *Proc. Int. Conf. Appl. Inf. Commun. (ICAIC)*, 2011, pp. 94–101.

[29] J. Shi, Y. Yang, and C. Tang, "Hardware assisted hypervisor introspection," *SpringerPlus*, vol. 5, no. 1, 2016, Art. no. 647.

[30] L. He *et al.*, "Dynamic secure interconnection for security enhancement in cloud computing," *Int. J. Comput. Commun. Control*, vol. 11, no. 3, pp. 348–357, 2016.

[31] M. A. Hakamian and A. M. Rahmani, "Evaluation of isolation in virtual machine environments encounter in effective attacks against memory," *Secur. Commun. Netw.*, vol. 8, no. 18, pp. 4396–4406, 2015.

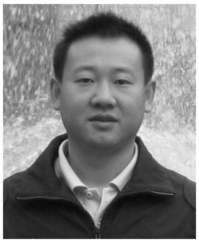
[32] J. Rushby, "Noninterference, transitivity, and channel-control security policies," Stanford Res. Inst., Menlo Park, CA, USA, Tech. Rep. CSL-92-02, 1992.



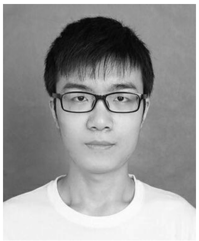
DELIANG XU received the bachelor's and master's degrees from the Hubei University of Technology, China. He is currently pursuing the Ph.D. degree with the School of Computer, Huazhong University of Science and Technology, China. His research interests include system security and wireless network security.

CAI FU (M'14) received the Ph.D. degree. He is currently an Associate Professor and a Deputy Director of the Information Security Institute, Computer School, Huazhong University of Science and Technology. His main research interests include wireless networking security, routing algorithms, and distributed computing.

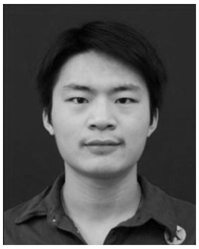
GUOHUI LI is currently a Professor with the School of Computer Science, Huazhong University of Science and Technology, China. His research interests include big data and data security.



DEQING ZOU received the Ph.D. degree. He is currently a Professor with the Computer School, Huazhong University of Science and Technology. His main research interests include trust computing, cloud computing security, networking security, routing algorithms, and distributed computing.



HONGHAO ZHANG received the B.E. degree in information security from the Huazhong University of Science and Technology, Wuhan, China, in 2016. He is currently pursuing the master's degree with the Huazhong University of Science and Technology, Wuhan, China. His research interests focus on cloud computing security and network security.



XIAO-YANG LIU received the B.A. degree in computer science and technology from the Huazhong University of Science and Technology, Wuhan, in 2010. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, Shanghai Jiao Tong University. His research interests include wireless communication, sensor networks, MANETs, VANETs, cyber physical system, and network security.