LMM
A Common Component for Software License Management on Cloud

Shinsaku Kiyomoto[1], Andre Rein[2], Yuto Nakano[1], Carsten Rudolph[2] and Yutaka Miyake[1]
[1]*KDDI R & D Laboratories Inc., Fujimino, Saitama, Japan*
[2]*Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany*

Keywords: Cloud Computing, Security, Software License Management, Digital Rights Management, Illegal Copy.

Abstract: On a cloud environment, the platform that runs a program is not fixed, and there is a possibility that a program runs on several servers in a cloud environment. Transferability of the license information by a valid user should be allowed, even though general requirements for license management have to be satisfied. In this paper, we consider software license management models for cloud environments, and discuss security functions for building secure license management schemes. We show four license management models and analyze the security requirements for the models. Then, we design a common component referred to as the license management module (LMM), and explain the security functions required for the LMM. Furthermore, we discuss how to realize the security functions and evaluate their performance using a prototype implementation.

1 INTRODUCTION

1.1 Background

Platform-as-a-Service (PaaS) is the service model of cloud computing, and it provides a program-executable environment for users. PaaS facilitates deployment of programs without the cost and complexity of buying and managing the underlying hardware and platform layers such as operating systems. However, some security risks of cloud computing services have been identified. It is impossible for users to verify the trustworthiness of all cloud computing environments, and there are concerns that operations in cloud computing may be carried out in an untrustworthy environment. Furthermore, a malicious user may be able to access the resources of other users, due to an architecture whereby users share resources on a cloud. The dynamic and fluid nature of the environments makes it difficult to maintain consistent security and ensure that records can be reliably audited. Thus, moving critical programs and sensitive data such as license information to a public and shared cloud computing environment raises major security concerns (Popovic and Hocenski, 2010; Shaikh and Haider, 2011; Kantarcioglu et al., 2011). In PaaS services, the platform provider supplies a platform for execution of user programs and users install or activate licensed programs on the platform. Users can access the platform and programs from a user terminal via the Internet. In this paper, we examine threats to cloud computing and in particular threats to installed licensed programs.

1.2 License Management on a Cloud

License checking using an activation code is a common function to protect software from being copied. Generally, the license is issued to a user who has the right to use the software, and the license allows the software to be used on a PC or other devices. If the user plans to use the software on two different environments, the user has to obtain two licenses for the software or register both environments to a license manager. Ownership of the licenses should be strictly managed and the transfer of the license from the user to other users has to be prevented by using a license management scheme. When a user executes a licensed program, license information should be securely stored and checked. On a cloud environment, the platform that runs a program is not fixed, and the platform is selected from cloud servers. There is a high possibility that a program runs on several servers in a cloud. Thus, transferability of the license information by a valid user must be allowed, even though general requirements for license management have to be satisfied. Furthermore, as the cloud servers are managed by a third party, some specific security

Kiyomoto S., Rein A., Nakano Y., Rudolph C. and Miyake Y..
LMM - A Common Component for Software License Management on Cloud.
DOI: 10.5220/0004504502840295
In *Proceedings of the 10th International Conference on Security and Cryptography* (SECRYPT-2013), pages 284-295
ISBN: 978-989-8565-73-0

requirements for cloud environments should also be considered.

Currently, four possible approaches are presented in ISO/IEC JTC1 SC7 WG21 as for ISO/IEC 19770 - Information Standard about Software Asset Management (SAM) (Bicket, 2011). These comprise the sourcing approach, contractual approach, structured approach, and mash-up approach. The sourcing approach provides free and open-source software, and the contractual approach allows license management to be delegated to cloud operators. The structured approach is based on the development and exploitation of a common technological architecture for software license management. The mash-up approach is based on a mix of different tools and approaches. The structured approach is the most efficient approach. A conceptual architecture for the technical mechanisms associated with asset management has been presented; however, the actual design of the architecture is still under consideration.

1.3 Issues Addressed in This Paper

PaaS service has a separation mechanism for each user environment to protect user resources. However, we should consider incorrect configuration of the hypervisor and some bugs in the separation mechanism. Even if the hypervisors are configured correctly, the risk of data intermingling or loss is still present (Courtney, 2012). Zhang *et al.* demonstrated an attack (Zhang et al., 2012) whereby a program on a virtual machine can steal a private key used by another virtual machine on the same server. We assume that a malicious user may access the resources of other users, and we also have to consider the threats posed by a malicious platform provider. In license management, we have to consider illegal actions by a licensed user who uses a client terminal and the cloud, in addition to malicious users and malicious platform providers on the cloud. The platform provider honestly executes user requests and cannot avoid performing any operations related to the programs. However, the platform provider may try to use the user's program maliciously or to obtain license information from physical memory or data storage. This model is a reasonable model where we consider the system manager of the platform as an attacker. It is a basic requirement for a software license management scheme to protect the software from illegal use and illegal copying by malicious users.

Thus, we should consider the following threats to secure cloud computing.

- Malicious users or malicious platform providers may access a program and execute it on the cloud platform without valid license information.
- Malicious users or malicious platform providers may steal license information from data stored on the cloud and/or the user terminal.
- External attackers try to steal license information from communication data, or alter communication data to execute a program on the cloud.
- Malicious users or malicious platform providers may copy a program and license information in order to use the program in a way that violates the scope permitted by the license.

Under the above threats model, a license management architecture for cloud environments should be considered.

1.4 Contribution of this Paper

To overcome the threats, we introduce a scheme using a License Management Module (LMM) which is capable of reducing the theft or misuse of licenses in general. Due to the application of memory protection schemes and other cryptographic methods, we will show how to significantly reduce threats for licensed programs in PaaS-environments. We present four license management models and analyze the security requirements for the models. Then, we design a common component LMM, and explain the security functions required for the LMM. Furthermore, we discuss how the security functions can be realized.

The rest of the paper organized as follows; Section 2 presents license management models, and security analyses. The design of the license management module and security functions is described in section 3. Evaluation results including security analysis and performance evaluation are presented in section 4, and existing research related to this paper is outlined in section 5. Finally, we conclude this paper in section 6.

2 LICENSE MANAGEMENT MODEL

In this section, we consider license management models for cloud environments. We assume a simple license management model in which a user license is issued for each application program of the user and only one program execution is allowed by each license information. A user cannot execute the program on two or more environments simultaneously, even though the user has valid license information. Frequent license checking is required, because a malicious user may move license information to another

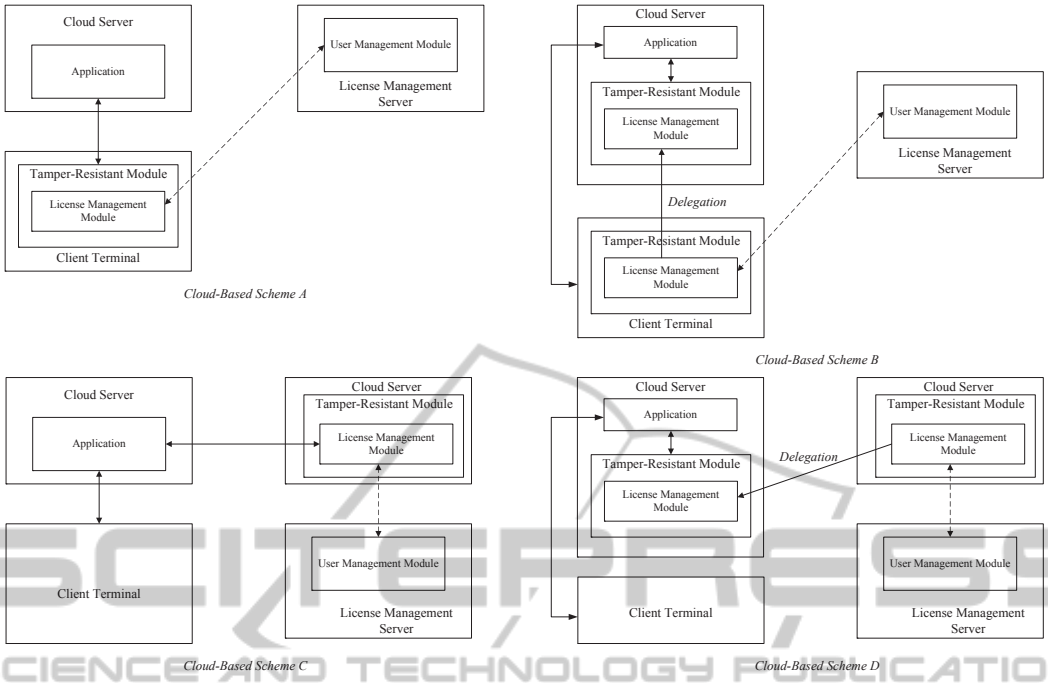

Figure 1: License Management Models.

environment after the license checking. First, we introduce three components for license management. The license management models are then built using these three components. Next, we present four license management models and discuss the advantages and disadvantages of the four models. Afterwards, we analyze security threats to the four license management models.

2.1 Components for License Management

License management is an important function for protecting software and contents against their illegal use. Generally, a user registers information to a software provider and receives in return the license information. We model license management schemes using three basic components as follows:

- *User Management Module.* The user management module manages the registration information of users. Users access the user management module to obtain license information before they start using the software. The user management module authenticates users, registers user information, and provides license information to users. We do not discuss the design of the user management module in this paper.
- *License Management Module.* The license management module (LMM) holds the license of a user. The LMM provides license information, when an application program requests the information from the LMM. The main issue of this paper is to examine the design of the license management module.
- *Application Program.* The application program is a program that a user uses with the license information. The application program is not executable without license information. The program accesses the LMM to obtain license information as soon as the program is executed. The application program may include digital contents such as musics and videos. In such cases, the application program is a viewer of the digital contents and a license management scheme for the application program is a part of a digital rights management (DRM) system for the contents. We discuss a general license management framework for application programs, and do not consider specific requirements for each program.

2.2 Cloud-based License Management Models

Four models are considered for license management on a cloud as shown in Figure 1. Models are separated based on the conditions under which a license management module (LMM) is implemented. In model A, the LMM is implemented on a user terminal, and

an application program communicates with the user terminal to check license information. Model B implements the LMM on both the cloud server and user terminal, and the license information is delegated to the LMM on the cloud. The LMMs in model C and model D are implemented on another cloud server that manages the license information.

We analyze the advantages and disadvantages of the four models as follows:

- **Model A.** Model A is an online scheme by which the license information is managed by the LMM on the user terminal, and a program running on the cloud accesses the LMM via a public network when the program is executed.
  Advantage. An advantage of model A is that no license information is stored in the cloud.
  Disadvantage. Frequent communication is required between the cloud server and the user terminal, where the license information is verified.
- **Model B.** Two LMMs communicate before the execution of a program in order to transfer license information. The LMM has to manage the delegated license information securely on the cloud.
  Advantage. No communication with another entity is required for license checking, even though communication for delegation of license information is needed.
  Disadvantage. The LMM and license information is on the cloud. We have to consider attacks on the cloud.
- **Model C.** This model is an extension of model A. The LMM is executed on another cloud server, and the LMM and a program communicate via a public network, when the program is executed.
  Advantage. No communication with the user terminal is required, although communication with another cloud is needed. We do not consider threats to the user terminal during license checking.
  Disadvantage. The cloud that holds the LMM should be trusted, otherwise we have to consider a protection mechanism for the LMM.
- **Model D.** This model is a combination of model B and model C. The LMM is executed on a cloud server, and another LMM on another cloud server receives delegated license information from the LMM that manages license information when the program is executed.
  Advantage. No communication with the user terminal is required, even though communication for delegation of license information is needed. We do not consider threats to the user terminal.
  Disadvantage. The cloud that holds the LMM should be trusted, otherwise we have to consider a protection mechanism for the LMM on both servers.

Sufficiency of License Management Models. We considered several license management models based on the LMM and selected the above four models. The models are separated by the location of the initial storage of the license information. Feasible options for the location are on the cloud or on the user terminal. As a consequence, there are models where the home LMM is on the user terminal and models where the home LMM is on the cloud server. The license information has to be stored in a unique LMM; however, it has to be possible to install and execute a program on several cloud servers. Thus, we cannot directly store license information in the cloud on which the program is executed, before the execution of the program. We also consider on-line or off-line license confirmation schemes for each model. We therefore examine the four license management models for a cloud environment. It should be noted that one can assume that the cloud server, that manages user LMMs, is trustworthy. In this case, it would be a simple license management model under a trusted license server, but this is beyond the scope of this paper.

2.3 Threats to License Management Models

In this section, we analyze possible threats to four license management models based on the threats discussed in 1.3. We have to consider three malicious entities: malicious user, malicious operator, and external attacker. The execution of a program without license information must be considered in all models. Illegal use of license information is another threat which may be executed by a malicious user and/or a malicious operator. All models except model A are vulnerable to this threats. These malicious entities may access the LMM and use the license information stored in the LMM. Model A does not consider attacks on the LMM on the cloud, even though attacks on communication between a user terminal and cloud server have to be considered. Model C does not have an LMM on the user terminal, and only considers attacks on the cloud and communication between cloud servers. Only local access to the LMM is used in model B, but communication for license delegation should be protected in this model. Threats to model D are similar to those for model C; so we assume threats to cloud servers. A summary of the analysis is shown in Table 1. The ”user” includes both attackers as a traitor and other users and ”operator” is a curious operator. Online schemes (such as model A) need not

Table 1: Threats to Each License Management Model.

Threats		Adversary	Target	Model A	Model B	Model C	Model D
Execution without License Info.		User, Operator	Program on Cloud	✓	✓	✓	✓
Illegal Use of License Info.		User, Operator	LMM on Cloud		✓	✓	✓
Theft of License Info. in	Terminal	User	LMM	✓	✓		
	Cloud	User, Operator	LMM		✓	✓	✓
Attack on Communication		User, Operator Ext. Attacker	Communication Data	✓	✓	✓	✓
Illegal copy of License Info. in	Terminal	User	LMM	✓	✓		
	Cloud	User, Operator	LMM		✓	✓	✓

be concerned about attacks on the LMM in a cloud; however, the drawback of the online schemes is that frequent communication is required for license confirmation, as discussed in 2.2. Offline schemes such as model B do not require communication while license confirmation is requested by the application program; however, we have to incorporate a protection mechanism for locally stored license information.

3 LICENSE MANAGEMENT MODULE

In this section, we design a common component for the license management module (LMM). We also present security functions for the license management models. In this paper, a cloud service provider is under a setting: *honest but curious* as discussed in 1.3. Thus, security and trust of the cloud environment are out of the scope of this paper.

3.1 Design of License Management Module

The License Management Module (LMM) is a key component for license management. We determined the requirements for the LMM according to the threats analysis in 2.3. The requirements for the LMM are summarized as follows:

- The LMM enables execution of the application program to be controlled according to license information.
- The LMM securely stores and manages license information.
- The LMM can securely transfer license information to other LMMs.
- Illegal copying of license information and the LMM has to be prevented.

We design the LMM based on the above requirements. We assume that the LMM is implemented on all cloud servers and client terminals. Figure 2 shows the design of the LMM. The module consists of five security functions: secure element, memory protection, application program binding, local binding, and delegation mechanism. The secure element stores secret information such as license information. The memory protection function protects data on physical memory against attacks such as a memory dump. The license information is loaded from the secure storage and stored in a physical memory using the memory protection function. The application binding function uses license information on the physical memory through the memory protection function and supports license confirmation with an application program. The application program does not complete any operation without help from the application binding function. That is, without a valid pair of the application program and license information, an attacker cannot execute the application program. The local binding function is used to protect the LMM against illegal copying. The function obtains local information that can identify an environment due to its uniqueness, and stores it in the secure element. The function also frequently obtains local information and compares it with the stored local information. If the information does not match, the LMM stops. The delegation mechanism is incorporated into model B and model D. The mechanism operates a delegation protocol between LMMs in order to transfer license information. Details of the functions are explained in the later sections. The software execution procedure is as follows:

1. When the program starts to execute, the program first accesses the LMM.
2. The LMM validates the identification information of a local environment using the local binding function.
3. If the identification information matches the information stored in the secure element, the LMM obtains the license information from the secure element, and stores it in the physical memory using the memory protection function.

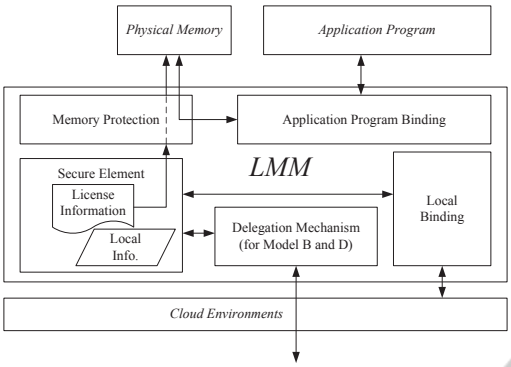

Figure 2: License Management Module.

4. The program frequently accesses the LMM to check license information, and the LMM provides the result of license check according to the application binding function.

3.2 Secure Element

The secure element is a form of secure storage that holds the license information and related secret values. The secure element is a part of the LMM, and can only be accessed by the LMM. The secure element can be implemented on both software and hardware. A tamper-resistant module is used for hardware implementation, and an encryption scheme is used for constructing a software-implemented secure element. In software implementations, a secret key for the encryption scheme needs to be securely embedded in program code of the LMM using software obfuscation. In the software implementation, encryption is not sufficient to protect license information, and how to restrict copying of the license information should also be considered. A status is defined for each license information[1] to avoid *double use* of the license file, and it should be protected. In consequence, an encrypted file including the status information has to be hidden from an operating system or it should be made difficult to find the files. Once the secret key is revealed, the LMM has to be replaced by a new one; the secret key should be embedded in the LMM securely. The hardware implementation is more secure than the software implementation, but the software implementation is widely applicable to several cloud environments.

Encryption Key Generation. Each license information should be encrypted by an encryption key derived from both a secret key and the user secret in order to avoid illegal use of license information. The user secret (for example a password) is provided by the owner of the license information, and it is requested when license information is used or delegated. When license information is stored in the secure element, the secure element request entry to the user secret, and derives the encryption key $EK_i = h(secret\ \ key||user\ \ secret)$, where $x||y$ denotes a concatenation of data x and data y. Note that the function $h(x)$ is a hash value computed by a hash function h. For a decryption of the license information, the secure element requests entry to the *user secret* again, and computes the encryption key in the same manner.

[1]We define six types of status: normal, read, delegating, delegated, delegated-read, and removed. Detailed explanations for the status types are shown in later sections

3.3 Memory Protection

The license information can be read from the secure element and extracted to physical memory, where its status is ”normal” or ”delegated”. The status of the license information is changed to ”read” or ”delegated-read” to avoid *double use* of the license information, where the license information is read from the secure element. We assume that an adversary including a cloud operator cannot dump and analyze the whole physical memory, but may dump and analyze a part of the physical memory. If an adversary dumps a part of the physical memory, the license information might be revealed. However, a huge computational cost is required when we dynamically encrypt/decrypt all data on the physical memory, during the program execution. Thus, a lightweight memory protection mechanism is used for the LMM. We design the memory protection function based on a memory protection scheme (Nakano et al., 2012). Figure 3 shows the procedure of the mechanism. The license information is divided into small peaces and mixed with random data. By the random data addition, the length l_d of data is extended to $k_m l_d$, where k_m is a security parameter that is selected according to security and performance requirements. The memory protection mechanism mediates memory accesses to memory area $\mathcal{L}$ and generates fake accesses. It hides information on which memory area is accessed. The scheme considers the buffer $\mathcal{M}$ being implemented within the secure element or secure memory area. The table $\mathcal{H}$ is also stored within this boundary. If we do not assume such a secure memory area, the buffer $\mathcal{M}$ with the size of m and history table $\mathcal{H}$ with the size of l_h are kept in the unsecured memory area as with $\mathcal{L}$, and they should be encrypted. In this case, access to $\mathcal{M}$ (and $\mathcal{H}$) is made by reading and writing every data block in the buffers (which requires decryption and encryption of data). Thus, to find out whether ‘a’ is in $\mathcal{M}$,

we read/write all values; when replacing data blocks in $\mathcal{M}$, we again read/write all values. The security provided is the same; but the computation overhead is obviously increased. It is expected that a huge computational cost does not requires to compute it, due to the size of $\mathcal{M}$. It is dependent on buffer size $\mathcal{M}$ and $\mathcal{H}$, but these sizes are sufficiently smaller than the size of the whole memory.

Data including both protected data and randomly generated redundant data are loaded into memory. After loading data, m data blocks are copied into $\mathcal{M}$; the table $\mathcal{H}$ starts empty. Before the program starts to run, the scheme can operate by accessing a number of dummy data blocks to populate the history buffer. On the first access to a data block 'a' (either a $\texttt{read}(a)$ or $\texttt{write}(a,x)$), the scheme searches for 'a' in $\mathcal{M}$ and accesses $\mathcal{L}$ twice: if 'a' is in $\mathcal{M}$, the scheme replaces two random elements (not 'a') from $\mathcal{M}$ with two random *dummy* elements from $\mathcal{L}$ and we access 'a'; if 'a' is not in $\mathcal{M}$, two random elements from $\mathcal{M}$ are replaced by 'a' and one random *dummy* element from $\mathcal{L}$ (and 'a' is accessed). In both cases, the corresponding addresses of blocks being kicked out from $\mathcal{M}$ are written in $\mathcal{H}$. In subsequent calls, the scheme proceeds as follows:

1. If 'a' is in $\mathcal{M}$, we replace two random elements (not 'a') from $\mathcal{M}$ with a random element from $\mathcal{L}$ and a random element from $\mathcal{L}$ which had already been accessed (as recorded in the *history* buffer), and we access 'a'.
2. If 'a' is not in $\mathcal{M}$, and its address is in the history table, the scheme replaces two random elements from $\mathcal{M}$ with a random element from $\mathcal{L}$ and 'a' (as recorded in the *history* buffer). Note that $\mathcal{H}$ holds only addresses and data are stored in $\mathcal{L}$.
3. If 'a' is not in $\mathcal{M}$, and its address is not in the history table either, the scheme replaces two random elements from $\mathcal{M}$ with 'a' and a random element from $\mathcal{L}$ which had already been accessed (as recorded in the *history* buffer).

Every time the data blocks are kicked out from $\mathcal{M}$ to $\mathcal{L}$, data blocks are written in $\mathcal{L}$ taking their original position (as described by $\mathcal{E}$), and the addresses of those blocks are registered in the history table $\mathcal{H}$. As the program continues to access data blocks, the table may eventually become full. When this occurs, at each access we select at random ℓ_h elements from among the $\ell_h + 2$ elements[2]. The memory is reshuffled after a certain period of time. If stronger security is required, data should be encrypted, when the data are uploaded into memory, although additional computational cost is required.

[2]The current history elements and two new ones

3.4 Application Binding

The application binding function enables software protection in a way such that the program cannot run without license information. The program frequently requires the license information to execute the whole operation. For that requirement, we develop a application binding scheme based on Fukushima *et al.*'s software protection scheme (Fukushima et al., 2012) and apply it to our license checking. The scheme transforms a target program into a protected application program and an its license information that is used by the application program binding function.

The application program is distributed to cloud environments and the user management module issues the license information to users. The protected program is executed on the platform and only handles encoded data according to Fukushima's scheme. The encoded data are data sets transformed by a secret encoding rule in the license information. After receiving the encoded output, the application binding function checks the validity of the output. If it is valid, the function returns the decoded result to the protected program. The application program binding function stores all the encoding rules $E_1, E_2, \ldots, E_m$ decoding rule D_1, and non-trivial relation R_1. The non-trivial relation checks the validity of the response from the protected program, and the decoding rule decodes the execution result[3].

The function checks the validity of the response in order to eliminate modified responses from an attacker. Let v_i be a variable in the program. The function calculates the value of $R_1(v_1, v_2, \ldots, v_m)$ from the response. If $R_1(v_1, v_2, \ldots, v_m) = 0$ holds, the function decides that the response is valid. Otherwise, it decides that the query is modified. The function decodes the value of x_1 using decoding rule D_1, i.e., $x_1 = D_1(v_1, v_2, \ldots, v_m)$, if the response from the protected program is valid.

Non-trivial Relations. An $m \times n$ Boolean matrix A with rank n and an m-dimensional vector b is selected as non-trivial relations for license information. Each element of the matrix must be 0 or 1, and each element of the vector must be an l-bit constant. A relational equation using matrix A, vector b, and rotation amounts $s_1, s_2, \ldots, s_n, t_1, t_2, \ldots, t_m$ is as follows:

$$\begin{pmatrix} y_1 \lll t_1 \\ y_2 \lll t_2 \\ \vdots \\ y_m \lll t_m \end{pmatrix} = A \begin{pmatrix} x_1 \lll s_1 \\ x_2 \lll s_2 \\ \vdots \\ x_n \lll s_n \end{pmatrix} \oplus b.$$

[3]There is no special reason to select R_1 for checking. We can use other non-trivial relations $R_2, R_3, \ldots, R_{m-n}$ instead of R_1.

The encoding rules $(E_1, E_2, \ldots, E_m)$ are obtained by solving this equation for encoded variables y_1, y_2, ..., y_m: $y_1 = E_1(x_1, x_2, \ldots, x_n)$, $y_2 = E_2(x_1, x_2, \ldots, x_n)$, $\ldots y_m = E_m(x_1, x_2, \ldots, x_n)$. Then, we solve the above equation for target variables x_1, x_2, ..., x_n. We have decoding rules $(D_1, D_2, \ldots, D_n)$ such as: $x_1 = D_1(y_1, y_2, \ldots, y_m)$, $x_2 = D_2(y_1, y_2, \ldots, y_m)$, $\ldots, x_n = D_n(y_1, y_2, \ldots, y_m)$. n out of m encoded rules are used to find the decoding rules. Non-trivial relations $(R_1, R_2, \ldots, R_{m-n})$ such as: $R_1(y_1, y_2, \ldots, y_m) = 0$, $R_2(y_1, y_2, \ldots, y_m) = 0$, ..., $R_{m-n}(y_1, y_2, \ldots, y_m) = 0$, are obtained by replacing target variables x_i with $D_i(y_1, y_2, \ldots, y_m)$ in the unused $m - n$ rules. The LMM stores all the decoding rules for managing application programs and sends decoded results to a valid application program.

3.5 Local Binding

A local binding mechanism is a security function to protect LMMs from illegal copying. The local binding mechanism is built into the LMM, and verifies the identification information of the local environment that the LMM is executed on. The procedure of local binding is as follows;

1. As an initial setting, the LMM obtains identification information from the local environment and computes a hash value for the information using a hash function. The LMM securely stores the hash value in the secure element.
2. When license information is requested, the LMM obtains identification information and computes its hash value. Then, the LMM compares the computed value with the hash value stored in the secure element. If the hash values are not identical, the LMM terminates the provision of license information.

If an adversary copies the LMM to his/her own device, the LMM does not provide the license information due to inconsistencies in the identification information generated from local environments. We use another method for local binding when we cannot use a secure element to store the identification information. In that case, we modify the encryption key generation procedure in 3.2. The key generation function is modified as $EK_i = h(secret\ key || user\ secret || identification\ information)$. If the identification information is not valid, the correct key is not obtained.

3.6 Delegation Protocol

In model B and model D, license information delegation between LMMs is required. The license delegation needs a secure communication channel between LMMs, and the channel can be constructed using an authenticated-key exchange protocol. A public-private key pair and a public key certificate are securely stored in the LMM. An LMM can verify the public key certificate and generate a session key from its own private key, a public key of the other entity, and random values that have been shared in the authenticated key exchange protocol. The status of license information should be managed in four stages: normal, read, delegating and removed. We also use an additional status "delegated" for delegated license information. The procedure of license delegation is as follows;

1. An LMM (LMM-1) obtains license information from the secure element. The status of the license information is changed from "normal" to "read".
2. LMM-1 executes an authenticated key exchange protocol with another LMM (LMM-2) that is the target to delegate the license information and construct a secure channel between the LMMs.
3. LMM-1 transfers the license information to LMM-2 via the secure channel, and the status is changed to "delegating". Then the status is changed to "removed" when the transmission is finished. The session key that LMM-1 holds is deleted.
4. LMM-2 stores it in its secure element. The label of the license information is "delegated". The session key that LMM-2 holds is deleted.
5. LMM-2 provides "delegated" license information for an application program during execution of the program.
6. After execution of the application program, LMM-2 computes a hash value of the "delegated" license information, and changes the status from "delegated" to "removed".
7. LMM-2 executes the authenticated key exchange protocol with LMM-1; then, LMM-2 sends a confirmation with the hash value of the license information to LMM-1 via a secure channel. LMM-2 removes the hash value.
8. LMM-1 checks the hash value and then LMM-1 changes the status of the license information from "removed" to "normal". LMM-2 removes the license information.

Note that the license information can be used where the status is "normal" or "delegated", and it can be delegated only when the status is "normal".

Table 2: Evaluation Results.

Function	PC	Cloud
Read License	43 ms	42 ms
Check License	37 ms	37 ms
Check Local Binding	36 ms	36 ms
Delegate License (except AKE)	83 ms	156 ms

The status is changed to ”delegated-read” when delegated license information is read from the secure element. We use an authenticated key exchange protocol based on Diffie-Hellman key agreement (Diffie and Hellman, 1976) for sharing a session key such as the ephemeral DH with RSA certificates mode (DHE-RSA) in TLS (Dierks and Rescorla, 2008). The trustworthiness of LMMs is confirmed based on their public key certificates in the protocol.

4 EVALUATION

In this section, we present evaluation results for the performance of the LMM and a security analysis of our license management models.

4.1 Performance Evaluation

We implemented a prototype system of the LMM on a PC[4] and a cloud environment[5], and evaluated the transaction time of each function. The total code size of the LMM was 314 KByte. Table 2 shows the evaluation results. All functions includes transaction time for using the memory protection. The transaction time of license delegation except the AKE protocol was evaluated on the PC and the Cloud. The total transaction time of license delegation depends on the selection of the AKE protocol. Each transaction time is sufficiently small. The transaction time of the memory protection linearly increases according to increase of the parameter k_m. We evaluated the transaction time where $k_m = 128$ and the overhead of the memory protection is negligibly small; thus, we can apply the LMM to practical services.

4.2 Security Analysis

When we apply the LMM to the four license management models, we are able to protect license information against the threats summarized in section 2.3, and prevent illegal use of application programs. In this subsection, we analyze the security of the license management in relation to the threats.

[4]Corei7-3720QM 2.6GHz, 8Gbyte Mem., CentOS 6.3

[5]Amazon EC2, Xeon 2.66GHz

Execution without License Information. The application binding function prevents program execution without license information because a key code is stored and executed on the LMM. As described, the license information is a vital part within the software protection scheme, as it contains information that needs to be present to successfully apply the encoding / decoding procedure within the software protection scheme. This means that the computational complexity of an attack on the software protection scheme without valid license information is $\Omega(exp(m\ log\ l))$ (Fukushima et al., 2012), where m is the number of variables involved in the software protection mechanism and l is the length of the variables.

Illegal Use of License Information. The license information is encrypted by an encryption key that is derived from a secret key of an LMM and user secret of the license owner. For a malicious user or operator this means that both the secret key and the user password need to be obtained before the license information can be used. As described, the user secret is never used directly in the encryption process. Instead a secure hash function is used to calculate a hash value from the user secret. By using a collision resistant secure hash function like SHA-256 or SHA-512, attacks on the user secret can nearly be eliminated. The only possibility would be to intercept the secret when it is entered by the user at the terminal. This has to be prevented by additional security mechanisms which are not considered in this paper. Consequently, a malicious user or malicious operator cannot use the license information without obtaining the user secret.

Theft of License Information. For this threat we have to distinguish between a running and a not running application. In particular, this means that either the license information is only available encrypted within the secure element (this is valid as long as the application is not executed) or the license information is already loaded into system memory (this is valid for an already executed application). So, as long as the application is not executed, we assume that the theft of license information additionally involves the theft of the user secret and access to the secret key. On runtime, the license information is already loaded into memory, and it is difficult for an attacker to determine correct data on the memory, even if the attacker can obtain δ access patterns to the physical memory. Due to (Nakano et al., 2012), the probability ε that the attacker finds the correct memory access is $\varepsilon \leq \frac{p_M}{(m-2)^2} + \frac{(1-p_M)p_H}{(m-2)^2} + \frac{(1-p_M)(1-p_H)}{2(m-2)}$, where $p_M \geq (\frac{m-2}{m})^\delta, p_H \geq (\frac{l_h}{l_h+2})^\delta$.

Attack on Communication. The communication channel between two LMMs must be protected using the secure delegation protocol. We assume that

a standardized protocol like TLS is used to generate a secure session key during the handshake. As described, the secure delegation protocol also includes digital signatures, which are used to authenticate the systems involved. This means that we assume that an external attacker is not able to modify the transferred data undetected, and thus may misuse the license information. The only possibility for an external attacker is to eavesdrop the encrypted communication. Assuming here that standardized encryption algorithms like AES-128 are used, the security depends only on the confidentiality of the secret session key.

Illegal Copy of License Information. The local binding mechanism provides an LLM with protection against illegal copying by an attacker. As described, the security here relies on the application of a secure hash function on local environment information. As any access to the license information also involves the verification of that particular information, it is not possible to copy the LLM and use the license information on a system different from the original system. In the models, the license information is transferred securely from one LMM to another LMM.

5 RELATED WORK

In this section, we briefly introduce existing work related to software license management on cloud environments.

License Management. License checking using an activation code is a common function of copy protection software. Software obfuscation schemes transform the original program into an obfuscated program that is difficult to analyze, while preserving its function. Some of the obfuscation schemes focus on obscuring the data structures in a program (Collberg et al., 1997; Shokurov, 2004), and some focus on obscuring the control flow (Hohl, 1998; Wang et al., 2001; Chow et al., 2001). Software watermarking schemes embed auxiliary information for identification into a program (Monden et al., 2000; Collberg and Thomborson, 2002). The drawback of watermarking is that it cannot prevent or detect unauthorized copying in real time. Online schemes use an external server to check the user license (Microsoft Corporation, 2003; Dvir et al., 2005) or to execute essential parts of a program (Zhao et al., 2009; Mumtaz et al., 2005). The drawbacks of these schemes are that the program manufacturer must deploy and manage a server, and program execution requires network access between the server and user terminal. Hardware-based schemes execute a whole program on a special hardware device, such as a secure processor (Gilmont et al., 1998; Suh et al., 2003; Shi et al., 2006). These schemes are expensive solutions since tamper-proof, high-performance hardware is needed to execute an entire program in a protected domain on the hardware. Combined hardware and software schemes reduce the deployment cost of hardware-based schemes. A program is executed on an unprotected device and only the important functions are executed on resource-constrained hardware. Atallah and Li (Atallah and Li, 2003) proposed a license management scheme for smart cards. In this scheme, the program is unprotected on the device. Mana and Pimentel (Mana and Pimentel, 2001) and Zhang and Gupta (Zhang and Gupta, 2003) presented schemes where only the essential parts of the program are executed on secure hardware. Their schemes require the development of distinct functions on each piece of hardware. Anderson (Anderson, 2008) studied the theoretical aspects of the combined schemes based on complexity theory. A program can execute securely in conjunction with a partnering oracle on a secure device. The oracle collaborates with any program by changing the parameters, since the function of the oracle is common to all programs. Anderson proved that Turing machines are secure under the assumption that a secure device can be used. Fukushima et al. (Fukushima et al., 2012) proposed a practical software protection scheme that transforms a general program into a protected program. The protected program handles encoded data on an unprotected device; then, the TPM decodes the execution result. Oblivious RAM is the traditional solution for memory access pattern protection. It was first proposed by Goldreich (Goldreich, 1987), and later extended by Goldreich and Ostrovsky (Goldreich and Ostrovsky, 1996). Oblivious RAM constructions remain too expensive to be implemented on embedded processors. In (Zhuang et al., 2004), Zhuang *et al.* proposed a practical, hardware-assisted scheme for embedded processors, with low computational overhead.

License Management on Cloud. Two general models have been developed in the European projects BEinGRID and SmartLM for managing software licenses in distributed environments (Li et al., 2008). The SmartLM project (The SmartLM Project, 2013) has addressed the licensing problem by working on a framework. The SmartLM project proposed a model that makes licenses mobile objects. The main approach of the SmartLM is to provide platform-independent access and treat software licenses as services. In the model, a license management service is used as a central service for license administration, license storage and scheduling as well as license information distribution and usage record cre-

ation. The model is similar to the cloud-based scheme (Model C) in our paper, and it can be used for license management on the cloud, even though the license management server needs to be a trusted local server. ElasticLM (elasticLM, 2010) is a license management scheme based on a contribution from the SmartLM project, and it is a novel technology for creating and managing software licenses designed for distributed computing environments like grids, clouds or service-oriented architectures (SOAs). The BeinGRID project provides a license tunneling model for FlexLM-based products (Simmendinger et al., 2008; The BEinGRID Project, 2009). Our LMM will be applicable to their architectures. GenLM (Dalheimer and Pfreundt, 2009) is a license management framework that allows software venders to manage their license usage in a distributed environment. The key idea of the framework is to build an on-demand license for a given job. Raekow *et al.* presented a license management architecture (Raekow et al., 2010) that allows the authentication of a job via a trusted proxy server. The architecture enables pay-per-use license management that can be deployed together with an on-demand computing scenario. Hou *et al.* designed and implemented a software license management scheme (Hou et al., 2007) for a campus computational grid environment; the main approaches are automatic license scheduling based on application queries, dynamic and license monitoring. Goel *et al.* presented a short paper (Goel and Dua, 2012) about a framework for a license management system between the cloud user, cloud vender, and cloud provider. All architectures/frameworks are assumed to be forms of online license management; thus, offline license management such as cloud-based scheme (B) and (D) in this paper have not been considered.

6 CONCLUSIONS

In this paper, we considered license management models for cloud environments and designed a license management module (LMM) for the models. We also considered security functions in the LMM. Evaluation results showed that the performance of the LMM is sufficiently feasible, and the LMM is applicable to license management on clouds. We will consider a license management mechanism for more complex licenses such as group licenses and hierarchical licenses in our future research.

ACKNOWLEDGEMENTS

A part of this work was supported by the Japanese Ministry of Internal Affairs and Communications funded project, "Study of Program Protection Mechanism for Open Platform Architectures (PROPROM)."

REFERENCES

Anderson, W. E. (2008). On the secure obfuscation of deterministic finite automata. In *Cryptology ePrint Archive, 2008/148*.

Atallah, M. J. and Li, J. (2003). Enhanced smart-card based license management. In *Proc. of IEEE International Conference on E-Commerce (CEC2003)*.

Bicket, D. (2011). Cloud computing and license management. *ISS-N004-v3.2*.

Chow, S., Gu, Y., Johnson, H., and Zakharov, V. A. (2001). An approach to the obfuscation of control-flow of sequential computer programs. In *Proc. of 4th Information Security Conference (ISC2001), Lecture Notes in Computer Science 2200*, pages 144–155.

Collberg, C., Thomborson, C., and Low, D. (1997). A taxonomy of obfuscating transformations. Technical Report 148, Computer Science, University of Auckland.

Collberg, C. S. and Thomborson, C. (2002). Watermarking, tamper-proofing, and obfuscation — tools for software protection. *IEEE Trans. on Software Engineering*, 28(8):735–746.

Courtney, C. (2012). Cloud computing security risks ? hypervisor and multi-tenancy. *Cloud Security*.

Dalheimer, M. and Pfreundt, F.-J. (2009). GenLM: license management for grid and cloud computing environments. In *Proc. of 9th ACM/IEEE International Symposium on Cluster Computing and the Grid (CCGRID '09)*, pages 132 –139.

Dierks, T. and Rescorla, E. (2008). The transport layer security (TLS) protocol version 1.2. *Internet Engineering Task Force (IETF), RFC5246*.

Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644 – 654.

Dvir, O., Herlihy, M., and Shavit, N. N. (2005). Virtual leashing: Internet-based software piracy protection. In *Proc. of 25th IEEE International Conference on Distributed Computing Systems (ICDCS2005)*.

elasticLM (2010). elasticLM - License as a Service (LaaS). *The Fraunhofer Institute for Algorithms and Scientific Computing SCAI*.

Fukushima, K., Kiyomoto, S., and Miyake, Y. (2012). Software protection combined with tamper-proof device. *IEICE Trans. on Fundamentals*, E95-A, No.1:213–222.

Gilmont, T., Legat, J.-D., and Quisquater, J.-J. (1998). An architecture of security management unit for safe hosting of multiple agents. In *Proc. of the International Workshop on Intelligent Communications and Multimedia Terminals*.

Goel, U. and Dua, R. L. (2012). A review paper on cryptographic approach for license management system in cloud computing. In *Indian Journal of Computer Science and Engineering (IJCSE)*, volume 3, No.4, pages 626–631.

Goldreich, O. (1987). Towards a theory of software protection and simulation by oblivious rams. In *Proc. of* ACM STOC *'87*, pages 182–194.

Goldreich, O. and Ostrovsky, R. (1996). Software protection and simulation on oblivious rams. In *Journal of ACM*, volume 43(3), pages 431–473.

Hohl, F. (1998). Time limited blackbox security: Protecting mobile agents from malicious hosts. In *Lecture Notes in Computer Science 1419*, pages 92–113.

Hou, Z., Zhou, X., and Wang, Y. (2007). Software license management optimization in the campus computational grid environment. In *Proc of the third International Conference on Semantics, Knowledge and Grid*, pages 604 –605.

Kantarcioglu, M., Bensoussan, A., and Hoe, S. (2011). Impact of security risks on cloud computing adoption. In *2011 49th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 670 –674.

Li, J., Weldrich, O., and Ziegler, W. (2008). Towards sla-based software licenses and license management in grid computing. In *From Grids to Service and Pervasive Computing*, pages 139–152.

Mana, A. and Pimentel, E. (2001). An efficient software protection scheme. In *Proc. of 16th IFIP International Conference on Information Security (ISC2001)*, pages 385–401.

Microsoft Corporation (2003). Technical overview of windows rights management services for windows server 2003.

Monden, A., Iida, H., Matsumoto, K., Inoue, K., and Torii, K. (2000). A practical method for watermarking java programs. In *Proc. of 24th Computer Software and Applications Conference (COMPSAC2000)*, pages 191–197.

Mumtaz, S., Iqbal, S., and Hameed, E. I. (2005). Development of a methodology for piracy protection of software installations. In *Proc. of International Multitopic Conference (INMIC2005)*.

Nakano, Y., Cid, C., Kiyomoto, S., and Miyake, Y. (2012). Memory access pattern protection for resource-constrained devices. In *Proc. of The 8th Smart Card Research and Advanced Application Conference (CARDIS2012), LNCS*, volume 7771, pages 188–202.

Popovic, K. and Hocenski, Z. (2010). Cloud computing security issues and challenges. In *MIPRO, 2010 Proceedings of the 33rd International Convention*, pages 344 –349.

Raekow, Y., Simmendinger, C., Grabowski, P., and Jenz, D. (2010). License management in grid and cloud computing. In *Proc. of 2010 International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*, pages 9 –15.

Shaikh, F. and Haider, S. (2011). Security threats in cloud computing. In *2011 International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 214 –219.

Shi, W., Hsien-Hsin, Falk, S. L. L., and Ghosh, M. (2006). An integrated framework for dependable and revivable architectures using multicore processors. In *Proc. of the 33rd International Symposium on Computer Architecture (ISCA2006)*.

Shokurov, A. (2004). An approach to quantitative analysis of resistance of equivalent transformations of algebraic circuits. Technical report, Institute for System Programming Russian Academy of Sciences.

Simmendinger, C., Kraemer-Fuhrmann, O., and Raekow, Y. (2008). Support for client-server based license management schemes in the grid. In *Collaboration and the Knowledge Economy: Issue, Application, Case Studies*, pages 1262–1272.

Suh, G. E., Clarke, D., Gassend, B., van Dijk, M., and Devadas, S. (2003). Aegis: Architecture for tamper-evident and tamper-resistant processing. In *Proc. the 17th annual international conference on Supercomputing*, pages 160–171.

The BEinGRID Project (2009). BEinGRID, business experiments in grid. *the European Unionfs sixth research Framework Programme (FP6)*.

The SmartLM Project (2013). SmartLM - grid-friendly software licensing for location independent application execution. *the European Commission Programme, Information and Communication Technologies*.

Wang, C., Davidson, J., Hill, J., and Knight, J. (2001). Protection of software-based survivability mechanisms. In *Proc. of International Conference of Dependable Systems and Networks (DSN2001)*, pages 193–202.

Zhang, X. and Gupta, R. (2003). Hiding program slices for software security. In *Proc. of the international symposium on Code generation and optimization: feedback-directed and runtime optimization*, pages 325–336.

Zhang, Y., Juels, A., Reiter, M. K., and Ristenpart, T. (2012). Cross-vm side channels and their use to extract private keys. In *Proc. of the 2012 ACM conference on Computer and communications security*, CCS '12, pages 305–316.

Zhao, J., Yao, N., and Cai, S. (2009). A new method to protect software from cracking. In *Proc. of World Congress on Computer Science and Information Engineering (CSIE2009)*, pages 636–638.

Zhuang, X., Zhang, T., Lee, H.-H. S., and Pande, S. (2004). Hardware assisted control flow obfuscation for embedded processors. In *Proc. of* ACM CASES *2004*, pages 292–302.