A Design of Mobile Trusted Module for Application Dedicated Cryptographic Keys

Daewon Kim, Yongsung Jeon, and Jeongnyeo Kim
Cyber Security Research Department
Electronics and Telecommunications Research Institute
Daejeon, Korea
emails: {dwkim77, ysjeon, jnkim}@etri.re.kr

***Abstract*—Normally, users encrypt data with cryptographic keys to protect original contents from various hackings. The use of cryptographic keys means that the protection of cryptographic keys is also an important problem as much as that of the encrypted data. A common way for protecting the keys is to authenticate user's key access authorities through some key passwords. However, nowadays the passwords can be easily exposed to a variety of password hacking techniques. The facts that the encrypted data is stored in unsafe storage, such as hard disk drivers or secure digital memory cards and that the cryptographic keys are accessed with any passwords mean that the encrypted original contents are no longer safe from the hackings. It is because hackers can decrypt user's encrypted data with the acquired passwords after they modify user's original applications or create new malicious applications. To solve this issue, we have developed a new mobile trusted module chip and management middleware based on the architecture with a key access mechanism dedicated to an application. In this paper, we present the design and operation of mobile trusted module chip and middleware together with some experimental results.**

Keywords-trusted platform module; mobile trusted module; hardware security module; integrity verification; cryptography.

I. INTRODUCTION

Data encryption is a common way to protect original contents from data hackings. For encrypting, some cryptographic keys and the authorities, which can be passwords for using the keys, are required. Normally, the encrypted data and keys are stored in some unsafe storage devices, which may be Hard Disk Drivers (HDD) or Secure Digital (SD) memory cards. Moreover, the passwords with authorities for accessing the keys can be easily exposed to various password hacking techniques. It means that the encrypted data can be decrypted by the malicious applications modified or created by hackers who already know the passwords of cryptographic keys. Finally, the traditional systems for managing cryptographic keys and passwords cannot guarantee the confidentiality of original contents included in the encrypted data from hackers.

From a few years ago, some hardware modules [1]-[5] have been used for encrypting data and for managing cryptographic keys. They are representatively Hardware Security Module (HSM) [1]-[4], Trusted Platform Module (TPM) [5], Mobile Trusted Module (MTM) [5], and so on. The hardware modules independently attached to user's devices include access control functions and require any authority information such as a password to access the critical data, which may be cryptographic keys, in the modules. Therefore, although hackers acquire the privileged authority of a target device itself, they cannot directly get important data in the modules.

The hardware modules can partially provide the safe storage for important data such as cryptographic keys and the access control functions commonly based on passwords. However, the reliability of password safety is gradually decreasing by a variety of password hacking techniques such as key hooking, screen capturing and social engineering methods. Additionally, if the password inputs are frequently requested, it can make normal users uncomfortable. In our previous work [9], we discussed some problems of password-based key access and proposed a mechanism verifying the integrity of application executed for accessing cryptographic keys in our MTM.

Another consideration of key management based on previous hardware modules is that the cryptographic keys can be accessed by each other applications. It means that malicious applications created by hackers can use the keys as well. Moreover, if several regular applications share a cryptographic key, due to a password exposed to any security vulnerability of an application the encrypted data of other applications sharing the key can also be at risk of information leakage. As the mobile work environment of Bring-Your-Own-Device (BYOD) is spreading more, the issue needs to be treated carefully.

As our considerations of the above problems, in this paper, we describe the design and operation of our prototype MTM and management software. The paper also includes a brief of our previous work related to the authority for accessing keys in the MTM. In the MTM, there are cryptographic keys dedicated to the authorized application and the application only can use the keys in the MTM. It means that hackers cannot use the keys to decrypt target encrypted data through the applications maliciously made by them although they know the passwords of keys. Additionally, our prototype basically supports the default cryptographic keys per application for users. Therefore, users do not need to create and manage the cryptographic keys with complex options for the simple and quick encryptions. It is sure that the users can

 ISBN: 978-1-61208-366-7

also create and manage the keys with various options suitable to their purposes.

The rest of paper is organized as follows; Section II represents the related works about hardware modules to be attached to user devices for securities. Section III describes the operations and features of our MTM architecture together with our previous work [9]. Section IV shows the detailed operations of our mechanism with some experiment results. Finally, Section V concludes this paper with a summary and future works.

II. RELARED WORKS

HSMs [1]-[4] are hardware modules including cryptographic hardware engines for specific services. They are commonly composed of cryptographic key generator, public key cryptography engine, symmetric key cryptography engine, the composition engine of two cryptographies, random number generator, and so on. Internal critical keys of HSMs do not be exposed to any outside and for accessing or using the keys, application users have to input any passwords to the HSMs. Normally, additional safe storage is not supported in them.

TPM and MTM [5] are hardware modules for user device security introduced by Trusted Computing Group (TCG), and TCG is documenting specifications for hardware modules and software stacks. HSM is a hardware module specified to a service, and otherwise TPM and MTM have the feature of common platform with the standardized Application Programming Interfaces (APIs) for data protection. They have an Endorsement Key (EK), which is a key embedded from the factory, and a Storage Root Key (SRK) is generated from the EK if a user gets the ownership of them. Other cryptographic keys are generated by the random number generator and cryptographic key generator in TPM and MTM, and after the keys are encrypted by key chains started from the SRK, the encrypted keys are stored into the file system of user device. Data encrypted by the cryptographic keys are confidential because the EK and SRK are not exposed to the outside of TPM and MTM. For using the keys that the platform of TPM and MTM manage, application users have to input passwords through TCG key management APIs.

There are a few researches [6]-[8] for simplifying the complex processes for accessing the keys in TPM and MTM. They provide wrapper APIs that integrate TCG APIs with commonly needed functions and minimize the effort that developers write applications with the standardized TCG APIs. However, user-friendly minimization of TCG APIs has a limit because its integration level is bounded under the basic hardware operations of TPM and MTM.

Our previous work [9] described a mechanism to authenticate an application for accessing the important data and keys in the new MTM designed by us. When an application with a certificate is installed into a user mobile device with the MTM, the integrity information of application is stored into the MTM. The application executed by a user is verified with the integrity information in the MTM and acquires the authority for accessing the application dedicated data in the MTM. The dedicated data such as cryptographic keys and private information can be accessed only by the verified application.

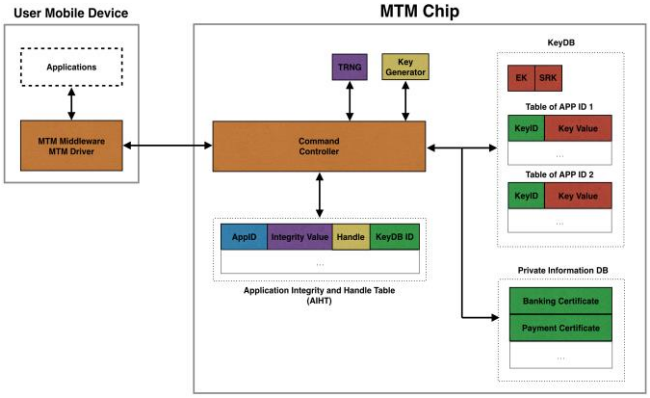

Figure 1. The overview of MTM architecture.

III. THE MTM ARCHITECTURE FOR APPLICATION DEDICATED CRYPTOGRAPHIC KEYS

A. Overview

Fig. 1 shows the architecture for presenting the operation mechanism of our MTM. The detailed mechanism for verifying application integrity has been described in our previous work [9]. By the previous work, if an application is regally installed in a user mobile device, the AppID and the measured integrity value of the application are stored into Application Integrity and Handle Table (AIHT) of our MTM and a default Storage Key (SK) of KeyID 0 is created in an APP ID Table of KeyDB. The SK is for the reserved key slot and is not used currently. The Table ID of KeyDB is also recorded to the KeyDB ID related to the integrity-verified application in AIHT.

When the installed application is executed by the user, the MTM middleware with an Integrity Measurement Agent [9] verifies the integrity of application and inserts an application-specific handle value randomly generated by the middleware into the MTM. The handle value is related to a communication channel between the middleware and the application. The middleware adds the handle value to the commands requested by the application and transfers them to the MTM. Through the handle value, MTM can access a table with APP ID in KeyDB and Private Information DB. Malicious applications cannot get a handle value from the middleware and cannot directly access to the MTM because they cannot know current handle values in the MTM.

B. Features of Our Prototype MTM

1) Integrity-based MTM Access Authority (IAA): If the integrity of application installed or executed by an user is successfully verified, Command Controller in our MTM inserts a handle value received from the MTM middleware into AIHT. It means that regal user applications can access some information, such as keys in the MTM without any passwords. Therefore, malicious or tampered applications

Figure 2. Experiment environment.

cannot access to the MTM due to none of handle values in MTM.

2) Application-dedicated cryptographic Key access Authority (AKA): The integrity-verified application has an authority for accessing a dedicated cryptographic key table in KeyDB. The middleware adds a pre-allocated handle value to every command messages received from the application. The application can access own keys by cryptography-related commands with the handle value. Therefore, applications cannot access keys of each other applications.

3) Default cryptographic Keys Ready (DKR): MTM prepares three default keys in the application-dedicated key table in KeyDB for supporting cryptography using MTM. If an application is verified and installed in the user mobile device, MTM automatically generates a SK in the key table. The KeyID of SK is zero, and it is first default key for reserving the zero of KeyID and is not used yet. KeyID is a index that the application accesses a key. Applications, normally, can use symmetric and asymmetric cryptographies, which are representatively Advanced Encryption Standard (AES) [10] and Rivest Shamir Adleman (RSA) [11]. If for the first time the application uses a cryptography command, our middleware and MTM processes the command after they automatically generates an symmetric or asymmetric key set. We defined the generated default keys with a 256-bit symmetric key of KeyID 1 and 2048-bit asymmetric keys of KeyID 2. Therefore, the application can simply use 256-bit AES and 2048-bit RSA, and it can also create new keys with user-defined KeyIDs.

IV. EXPERIMENTS

For verifying the cryptography feasibility of our MTM and middleware, we have experimented with a few examples in Android environment. Fig. 2 shows the experiment environment. In Fig. 2, a left upper part is the board that MTM chip has been mounted and a left lower is the interface board connecting user mobile device to the MTM board.

A. Specifications

For MTM, we use a smart card chip which includes CPU, OS, memory, and so on. The size of MTM chip is width 5 mm and height 5 mm. The MTM can support each 10 key sets about 300 dedicated applications except for normal applications. In this paper, we do not explain other detailed

```
public static class BindInfo
{
  int in_KeyID;
  int in_DataSize;
  byte[] in_Data;
  int out_DataSize;
  byte[] out_Data;

  public BindInfo()
  {
    // in_KeyID = 2;
    in_KeyID = MTM_KEYID_DEFAULT_ASYMMETRIC;
  }
}
```

(a)

```
MTM mtm = new MTM();
MTM_Crypto.BindInfo bind_info = new MTM_Crypto.BindInfo();

mtm.Open();

// Bind
bind_info.in_DataSize = 4;
bind_info.in_Data = new byte[] {
  (byte)0xaa, (byte)0xbb, (byte)0xcc, (byte)0xdd};
mtm.Bind (bind_info);

// User processing

// UnBind
bind_info.in_DataSize = bind_info.out_DataSize;
bind_info.in_Data = bind_info.out_Data;
mtm.UnBind (bind_info);

...

mtm.Close ();
```

(b)

Figure 3. The sample codes for estimating performance: (a) java class for Bind/UnBind and (b) the example code for Bind/UnBind.

- Kernel version: 3.0.15 SMP PREEMPT
- Android version: 4.0.4
- Core: ARM Cortex-A9 based Dual CPU
- Clock Speed: 1.2GHz
- Internal RAM: 128MB
- Memory: mobile DDR2 1GB, embedded Multi Media Card (eMMC) 16GB

Figure 4. Android mobile device specifications for experiments.

specifications of our MTM chip because the information is confidential yet. Fig. 4 shows user mobile device specification that applications are executed.

B. Experiment Scenario

For cryptography testing, we have experimented RSA encryption and decryption commands. We defined the commands as Bind and UnBind, which are same API names with TCG. The (a) and (b) of Fig. 3 shows real sample codes for Bind and UnBind. Like (b), users and applications do not need to manage keys for basic cryptography because the application dedicated default keys are supported through our middleware and MTM. Fig. 5 shows the message flows in (b) of Fig. 3 between an application and our MTM. The middleware adds a handle value on every command and sends a command for creating a default key to the MTM if there is not an appropriate key in the MTM.

C. Performance Estimation

1) Key Creation: We measured the elapsed time for creating a key set of RSA 2048 bits. As the experiments of 100 times, we presents the average time in Table I. Through the experiments, we also measured the overheads of our

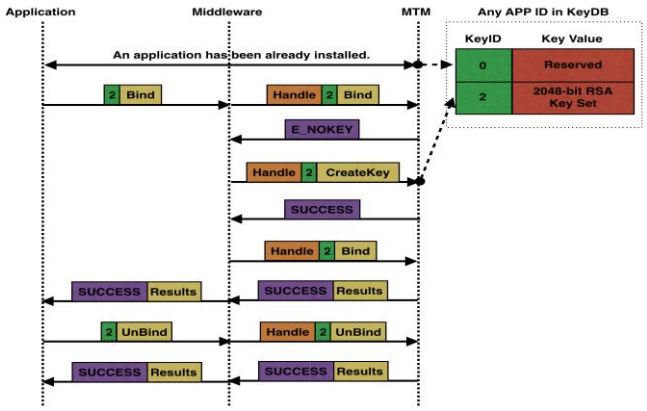

Figure 5. The message flows in (b) of Fig. 3. The middleware can support multiple applications communicating with MTM simultaneously.

softwares. In Table I, MTM Service Provider (MSP) is a client library linked to the application and MTM Core Service (MCS) is a service daemon with middleware functions. The time for generating RSA keys is variable because of the algorithm features of RSA key generation. We are trying to enhance the performance and the issue for waiting users will be treated as future work.

TABLE I. THE PERFORMANCE ESTIMATION OF CREATEKEY

Command Name	Round Trip Time (ms)		
	Application to MTM	*Driver to MTM*	*MSP+MCS Overhead*
CreateKey	5486	5485	1

2) Bind: We measured the elapsed time for encrypting the data of 1 and 214 bytes. In current, our MTM for RSA 2048 bits encryption with Optimal Asymmetric Encryption Padding (OAEP) mode can process maximum 214-byte data per one command message. As the experiments of 1000 times in Table II, we present the average Round Trip Time just from the application to MTM because the software overhead (MSP+MCS) 1 ms is a small part of full elapsed time.

3) UnBind: We measured the elapsed time for decrypting 256-byte data encrypted as RSA 2048 bits. As the experiments of 1000 times, we present the average time in Table II.

TABLE II. THE PERFORMANCE ESTIMATION OF BIND AND UNBIND

Command Name	Application to MTM RTT (ms)		
	1 Byte	*214 Bytes*	*256 Bytes*
Bind	88	116	-
UnBind	-	-	289

Currently, our prototype has about 30 commands including the above three commands.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we introduced the design and operation examples of our MTM and middleware. They have the features of Integrity-based MTM Access Authority (IAA), Application-dedicated cryptographic Key access Authority (AKA), and Default cryptographic Keys Ready (DKR). The features have the advantages that users can access the cryptographic keys, private information, and the secure operation of financial services user-friendly to the MTM. The comprehensive functions are not supported in other hardware security modules yet.

In current prototype, the time for generating RSA keys is too long to wait users when the default RSA key is used for the first time. In future work, we will create the default RSA keys at the time for installing an application instead of the time for using the keys. Additionally, the hardware logics for creating the RSA keys will be optimized.

ACKNOWLEDGMENT

This work was supported by the IT R&D program of MSIP/KCA. [12-912-06-001, Development of the Security Technology for MTM-based Mobile Devices and next generation wireless LAN].

REFERENCES

[1] T. Souza, J. Martina, and R. Custodio, "Audit and backup procedures for Hardware Security Modules," Proc. of the 7th symposium on Identity and trust on the Internet, 2008, pp. 89-97.

[2] B. Rosenberg, "Handbook of Financial Cryptography and Security," Chapman and Hall/CRC, 2010.

[3] J. Kang, D. Choi, Y. Choi, and D. Han, "Secure Hardware Implementation of ARIA Based on Adaptive Random Masking Technique," ETRI Journal, vol. 34, no. 1, Feb. 2012, pp. 76-86.

[4] M. Wolf and T. Gendrullis, "Design, implementation, and evaluation of a vehicular hardware security module," Proc. of the International Conference on Information Security and Cryptology (ICISC 2011), Springer Berlin Heidelberg, 2012, pp. 302-318.

[5] Trusted Computing Group. TPM main specification. Main Specification version1.2 rev116, Trusted Computing Group, March 2011.

[6] G. Cabiddu, E. Cesena, R. Sassu, D. Vernizzi, G. Ramunno, and A. Lioy, "The Trusted Platform Agent," IEEE Software, vol. 28, 2011, pp. 35-41.

[7] C. Stuble and A. Zaerin, "uTSS – A Simplified Trusted Software Stack," Proc. of the 3rd International Conference on Trust and Trustworthy Computing, 2010, pp. 124-140.

[8] R. Toegl, T. Winkler, M. Nauman, and H. Theodore, "Specification and standardisation of a java trusted computing api," Software: Practice and Experience, vol. 42, no. 8, 2012, pp. 945-965.

[9] D. Kim, J. Kim, and H. Cho, "An Integrity-Based Mechanism for Accessing Keys in A Mobile Trusted Module," Proc. of the International Conference on ICT Convergence, 2013, pp. 780-782.

[10] J. Daemen and V. Rijmen, "The Design of Rijndael: AES - The Advanced Encryption Standard," Springer, 2002.

 ISBN: 978-1-61208-366-7

[11] R. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” Commun. ACM, vol. 21, no. 2, 1978, pp. 120-126.

 ISBN: 978-1-61208-366-7