IEICE TRANS. INF. & SYST., VOL.E93–D, NO.5 MAY 2010

PAPER Special Section on Information and Communication System Security

Practical and Secure Recovery of Disk Encryption Key Using Smart Cards*

Kazumasa OMOTE[†a)], ***Nonmember and*** **Kazuhiko KATO**[††], ***Member***

SUMMARY In key-recovery methods using smart cards, a user can recover the disk encryption key in cooperation with the system administrator, even if the user has lost the smart card including the disk encryption key. However, the disk encryption key is known to the system administrator in advance in most key-recovery methods. Hence user's disk data may be read by the system administrator. Furthermore, if the disk encryption key is not known to the system administrator in advance, it is difficult to achieve a key authentication.

In this paper, we propose a scheme which enables to recover the disk encryption key when the user's smart card is lost. In our scheme, the disk encryption key is not preserved anywhere and then the system administrator cannot know the key before key-recovery phase. Only someone who has a user's smart card and knows the user's password can decrypt that user's disk data. Furthermore, we measured the processing time required for user authentication in an experimental environment using a virtual machine monitor. As a result, we found that this processing time is short enough to be practical.

key words: *user authentication, key recovery, smart card*

1. Introduction

Information leakage has recently become a serious problem. Because a user's disk might contain a lot of confidential information, it should be encrypted and the disk encryption key is protected securely. Hence, it is important not only to encrypt the disk data but also to store the disk encryption key securely. Disk security has been improved by storing the encryption key in a hardware token such as a smart card or USB device. We need to have some methods to recover the disk encryption key when the token is lost. For example, it is necessary to keep a backup copy in a safe place such as another key management server.

In key-recovery methods using smart cards, a user can recover the disk encryption key in cooperation with the system administrator, even if the user has lost the smart card including the disk encryption key. However, the disk encryption key is known to the system administrator in advance in most key-recovery methods. Hence user's disk data may be read by the system administrator. Furthermore, if the disk encryption key is not known to the system administrator in advance, it is difficult to achieve a key authentication.

Manuscript received July 31, 2009.
Manuscript revised December 18, 2009.

[†]The author is with Japan Advanced Institute of Science and Technology, Nomi-shi 923–1292 Japan.
[††]The author is with University of Tsukuba, Tsukuba-shi 305–8573 Japan.

*The preliminary version of this paper was presented at ITNG'08 [1].

a) E-mail: omote@jaist.ac.jp

DOI: 10.1587/transinf.E93.D.1080

In this paper, we propose a scheme which enables to recover the disk encryption key when the user's smart card is lost. In our scheme, the disk encryption key is not preserved anywhere and then the system administrator cannot know the key before key-recovery phase. Only someone who has a user's smart card and knows the user's password can decrypt that user's disk data. Hence, our scheme enables a user to not only prevent user's disk data from being stolen from user's PC but also recover his disk encryption key without his smart card. Furthermore, we measured the processing time required for user authentication in an experimental environment using a virtual machine monitor. As a result, we found that this processing time is short enough to be practical.

This paper is organized as follows. In Sect. 2 we briefly review some related works. We then in Sect. 3 explain our preliminaries, in Sect. 4 describe our scheme in detail, and in Sect. 5 present the results of an experimental evaluation of our scheme. In Sect. 6 we discuss our scheme and in Sect. 7 we conclude by briefly summarizing the paper.

2. Related Work

BitLocker Drive Encryption is a security feature integrated into the Windows Vista operating system for full disk encryption [2]. It works in combination with a TPM chip that encrypts the key used in encryption and saves the encrypted key on the disk. The Secure File System [3], on the other hand, provides filesystem-level encryption.

A lot of authentication schemes based on smart cards have been proposed recently [4]–[8]. In them a user's secret information that is shared with servers is stored in the user's smart card. It is protected by a password and by the difficulty of computing discrete logarithms. There is also a scheme that improves security by combining the use of a smart card and the Virtual Machine Monitor (VMM) [9].

Several papers about key recovery have been published [10]–[14]. Of the four kinds of key recovery methods (key escrow, trusted third party, commercial key backup and key encapsulation), key encapsulation is the only one in which the key is not known to the system administrator [11], [12]. When key encapsulation is used, however, it is hard to confirm that the recovered key is the legitimate user's key because the system administrator does not know the key in advance. This means that the encryption key can be recovered by a malicious user inside the system. Although we can easily consider a method that uses the key with a cer-

tificate, in that case the key would be known to the system administrator in advance.

Key-recovery methods using a smart card have been proposed [13], [14], but they are fundamentally different from the one in our scheme. Although the disk encryption key in [13] is not stored anywhere as well as our scheme, it does not consider the security against the system administrator, hence the system administrator can always compute the disk encryption key which is distributed to the smart card and the card reader. In [14], the smart card with the tamper-resistance is turned over to an escrow agent for safekeeping. These approaches are different from ours.

There is also a method in which secret keys are managed safely by using blind signatures and passwords [15]. In that method a user's secret key is encrypted by the value of the blind signature and the source of the signature is encrypted by the password. Both the encrypted key and the encrypted source are kept on a local disk. This double encryption protects the password from brute force attacks.

3. Preliminaries

3.1 Requirements

In this section, we describe some requirements of our scheme.

- **Weakened authority**. Although system administrators have high access authority, they should not know the disk encryption keys of users because they may not be authorized to read a user's sensitive information. Thus, it is important not to leak a private data to the system administrator.
- **Authenticated recovery of disk encryption key**. If the disk encryption key is known to the system administrator in advance, we can achieve a key authentication. Otherwise, it is difficult to achieve the key authentication. Although there are also some methods in which the user's disk encryption key need not be known to the system administrator in advance, with them it is difficult to ensure that a key can be recovered only by the legitimate user. We therefore want to conduct an authenticated recovery without letting the system administrator know the disk encryption key in advance.

3.2 Full Disk Encryption

There are two kinds of disk encryption: full disk encryption, in which all the byte data on the disk is encrypted; and filesystem-level encryption, in which what are encrypted are the files or directories on the disk [16]. We employ full disk encryption. An advantage of full disk encryption is that everything, including the swap space and the temporary files, is encrypted and the decision of which files to encrypt is not left to users [16]. Our scheme uses a symmetric-key encryption algorithm such as AES because it is a high-security algorithm and can encrypt/decrypt a disk quickly. In a full disk encryption, the OS is encrypted in a hard disk. So some program would start up the OS by decrypting the hard disk data.

3.3 Definition and Assumption

We define function to break the Computational Diffie-Hellman (CDH) assumption over $\mathbb{Z}_n$ [17].

Definition 1 (CDH over Z_n): CDH(n, g, A, B) is a function that on input $n \in \mathbb{N}_{>1}$, $g \in \mathbb{Z}_n^*$, $A \in \mathbb{Z}_n^*$, $B \in \mathbb{Z}_n^*$, outputs $C \in \mathbb{Z}_n^*$ such that $C = g^{ab} \bmod n$, where $A = g^a \bmod n$ and $B = g^b \bmod n$, if such a C exists.

Assumption 1: CDH over $\mathbb{Z}_n$ in Definition 1 is stronger than or equivalnet to the RSA.

4. Our Scheme

In this section, we explain the detailed protocol of our scheme.

4.1 Notations

In this description of the notations used in our scheme, $|n|$ and $|n_i|$ are assumed to be more than 1024 bits.

U_i	:	User i
pw_i	:	U_i's password
(e, d, n)	:	RSA keys of STTP
(e_i, d_i, n_i)	:	RSA keys of U_i
R	:	Public random number ($R \in \mathbb{Z}_n^*$)
r, r_i	:	Random numbers ($r, r_i \in \mathbb{Z}_n^*$)
a_i, b_i	:	Random numbers ($a_i, b_i < \|n\|/2$)
$cert_i$	:	Public key certificate of U_i
$CERT$	:	Public key certificate of STTP
CRL	:	Certificate revocation list
c	:	Challenge
sk_i	:	Disk encryption key of U_i
$h(\cdot)$	:	Cryptographic secure hash function (e.g., SHA1)

4.2 Entities

We explain three entities in our scheme (see Fig. 1)

- **STTP**. We assume a semi-trusted third party (STTP). The STTP stands for the system administrator and the trusted server (i.e. the STTP server). The STTP server is managed by the system administrator. The STTP sets up the scheme, registers a user U_i, issues $cert_i$ and

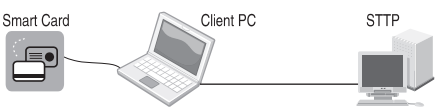

Fig. 1 Entities.

CRL, and recovers lost encryption keys. The STTP has user's information (user identity, $cert_i$, $R^{a_i b_i} \bmod n_i$ and $R^{d_i} \bmod n_i$), *CERT*, *CRL* and her RSA keys (e, d, n) on her own storage. Since the STTP is "honest-but-curious", the preserved user information may be leaked by the STTP.

- **Smart card**. A smart card is a hardware token which stores the disk encryption key. It is also an identification card such as an employee ID card or a student identification card. Hence the disk encryption key is linked to the user's identity. Thus, only a specific user who has the smart card is able to start up the OS in the user's client PC. The smart card is used for generating the key and for user authentication. Our scheme therefore generates the disk encryption key with the help of a smart card. We also assume that a PKI such as remote authentication is used with the smart card. The private key, the public key, the public key certificate, and the signature calculation software in the public key cryptosystem are stored in this card. User's RSA keys and user's certificate are stored in this card.
- **Client PC**. A client PC is used by a user and can connect directly with a smart card. While the user's OS is stored in an encrypted storage of a client PC, some information of $R^{a_i b_i} \bmod n_i$, $(a_i b_i)^e \bmod n$, *CERT* and *CRL* is stored in a non-encrypted storage of a client PC.

4.3 Premises

The premises of our scheme are these:

1. The STTP is "honest-but-curious": the STTP prevents the information stored in its database from being falsified, but may leak user's secret information such as the disk encryption key.
2. The STTP can freely alter the data in user's smart card by using the STTP's privileged password, but the STTP cannot read that data.
3. The STTP is not attacked.
4. The smart card is tamper resistant, and the confidential information is not stolen from the smart card itself or while it is being transferred between the smart card and the client PC.
5. The calculation algorithms (e.g., signature generation) in the smart card are not falsified.
6. The access password of the smart card is not cracked because the smart card is locked when a user input a wrong password more than the predetermined frequency.
7. A client PC has non-encrypted storage where some data and programs required for the OS booting are stored.
8. A client PC is not cracked by a malicious STTP. In other words, the STTP does not spy on any user's information in the client PC.
9. A client PC connected with the smart card which has completed user authentication is not stolen.
10. A user can read data other than his private key in his smart card by using his password, but he cannot alter any data except for his password.
11. A user has the private key only in his smart card.

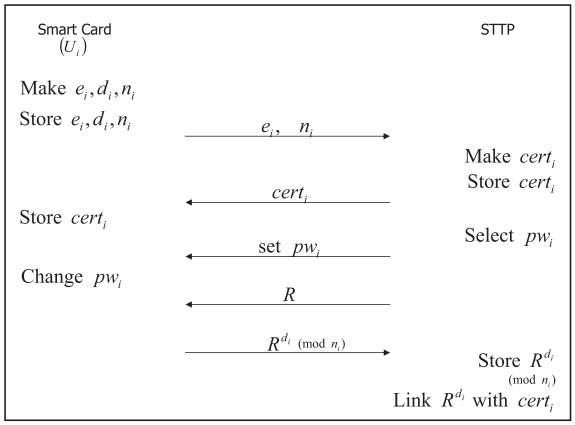

Fig. 2 Registration phase (1).

4.4 Protocol Description

The protocol in our scheme is composed of the following six phases.

- **System setup phase**. When the STTP starts our scheme, she executes this phase. The STTP generates its *RSA* keys, *CERT*, and *CRL*. It also generates the random numbers $R \in \mathbb{Z}_n^*$ that are the common public information of the system. A user installs some programs concerning the registration phase and the local-authentication phase to non-encrypted storage.
- **Registration phase (1)**. When the STTP registers a user, she executes the registration phases (1) and (2). The user U_i connects his smart card with the STTP server and initializes the card (see Fig. 2). At first, U_i makes his RSA keys (e_i, d_i, n_i) inside his smart card, and the smart card transfers his public keys (e_i, n_i) to the STTP. The value of (e_i, d_i, n_i) are stored in the smart card. Then, the STTP generates U_i's certificate $cert_i$ and returns $cert_i$ to the smart card. Both the STTP and U_i store the $cert_i$. The STTP sets the pw_i into the U_i's smart card and then the pw_i is changed by U_i. Finally, the STTP sends the public random number R to the U_i's smart card, and then the smart card computes the value of $R^{d_i} \bmod n_i$ and returns it. The STTP confirms the validity of $R^{d_i} \bmod n_i$ and stores it. Since this procedure is securely connected with the STTP directly, the R^{d_i} is linked with the $cert_i$ of U_i.
- **Registration phase (2)**. U_i directly connects the client PC to the STTP server, and makes information that is necessary for the key recovery (see Fig. 3). The communication between a client PC and the STTP is a secure channel using RSA signature in public key infrastructure. At first, the STTP sends *CERT*, *CRL* and R to the client PC. The client PC stores *CERT*, *CRL*

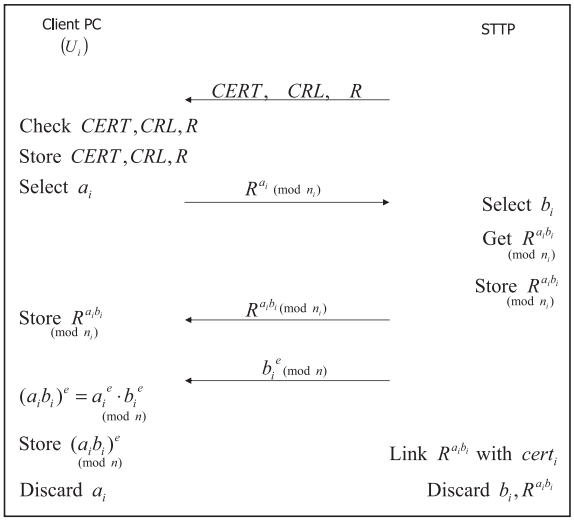

Fig. 3 Registration phase (2).

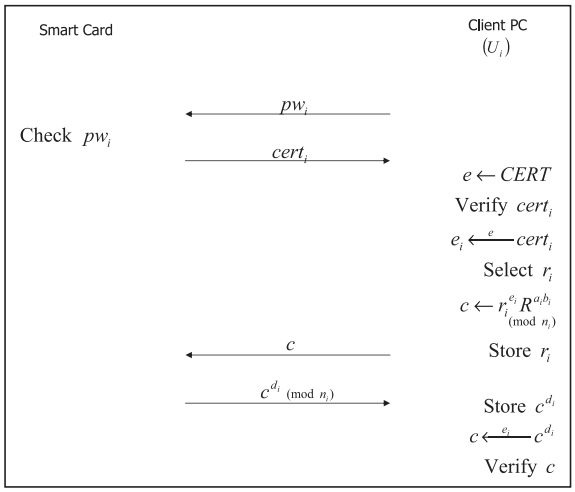

Fig. 4 Local-authentication phase.

and R in its own non-encrypted storage after it checks these values. Then, the client PC selects a random $a_i < |n|/2$, computes $R^{a_i} \bmod n_i$, and sends it to the STTP server. On the other hand, the STTP selects a random $b_i < |n|/2$, computes $(R^{a_i})^{b_i} \bmod n_i$, and sends it to the client PC. The client PC stores the $R^{a_i b_i} \bmod n_i$ in its own non-encrypted storage. Finally, the STTP server computes $b_i^e \bmod n$ and sends it to the client PC. The client PC computes $(a_i b_i)^e = a_i^e b_i^e \bmod n$ and also stores it in its own non-encrypted storage. The $R^{a_i b_i} \bmod n_i$ is linked with $cert_i$ because $R^{a_i} \bmod n_i$ corresponds to $cert_i$. The client PC uses the smart card initialized in the registration phase (1) to get $cert_i$. While the client PC discards a_i, the STTP server discards b_i and $R^{a_i b_i} \bmod n_i$. Note that neither the client PC nor the STTP server knows the value of $a_i b_i$.

- **Local authentication phase**. When a user uses a client PC, this local authentication phase and the following key-generation phase are executed before the OS booting. The client PC confirms the validity of the smart card in this phase. We can confirm that the smart card was legitimately issued by the STTP. The procedure is shown in Fig. 4. At first, the smart card authenticates U_i by his password pw_i. Only if the password is valid, the smart card sends $cert_i$ to the client PC. The client PC verifies $cert_i$ by e which is extracted from $CERT$. The client PC computes $c = r_i^{e_i} R^{a_i b_i} \bmod n_i$, where e_i is extracted from $cert_i$. Finally, the client PC sends c to the smart card as a challenge, and then the smart card computes $c^{d_i} \bmod n_i$ and returns it. The client PC verifies the value of c by checking $c \stackrel{?}{=} (c^{d_i})^{e_i} \bmod n_i$. In this phase we use "a blind signature" for both the generation of the disk encryption key and the authentication of smart card. Note that U_i preserves c^{d_i} and r_i to use in the next phase.
- **Key-generation phase**. When a local authentication succeeds, the client PC starts the OS by the following procedure.
  1. The client PC derives the legitimate disk encryption key sk_i of size $|n_i|$ by calculating $c^{d_i}/r_i \bmod n_i$:
  $$sk_i = R^{a_i b_i d_i} \pmod{n_i}. \quad (1)$$
  2. The client PC divides the disk encryption key sk_i by the block size of the symmetric-key encryption algorithm, and then uses some chopped keys from the head of byte data of sk_i, which is shown in practice as follows: $sk_i = sk_{i1} \| sk_{i2} \| \cdots$, where $\|$ denotes concatenation. The first chopped key is sk_{i1}. For instance, when the symmetric-key encryption algorithm is AES-128, you can obtain eight disk encryption keys because $|n_i| = 1024$ bits. In this case you use only the first key, sk_{i1}.
- **Key-recovery phase**. When you accidentally lose your smart card or your password, you execute the key-recovery phase by connecting your client PC with the new smart card to the STTP server directly (see Fig. 5). This new smart card is used for user authentication. U_i has to obtain the new smart card from a system administrator before starting key-recovery phase. If U_i loses his smart card, a new smart card needs to be reissued for him. On the other hand, if U_i forgets his password, he executes the registration phase (1) to initialize his smart card. Then, U_i has to execute the key-recovery phase using such new smart card. Here, the STTP possesses two kinds of U_i's certificates ($cert_i$ and $cert_i'$). The $cert_i$ stands for the previous certificate and the $cert_i'$ stands for the new certificate. Note that only the STTP has the previous certificate $cert_i$ as a valid certificate for key-recovery although $cert_i$ is already added to CRL. At first, the client PC sends the key-recovery request message to the STTP. Then, the STTP checks whether both $cert_i$ and $cert_i'$ exist in the STTP server. If two kinds of certificates exist, the

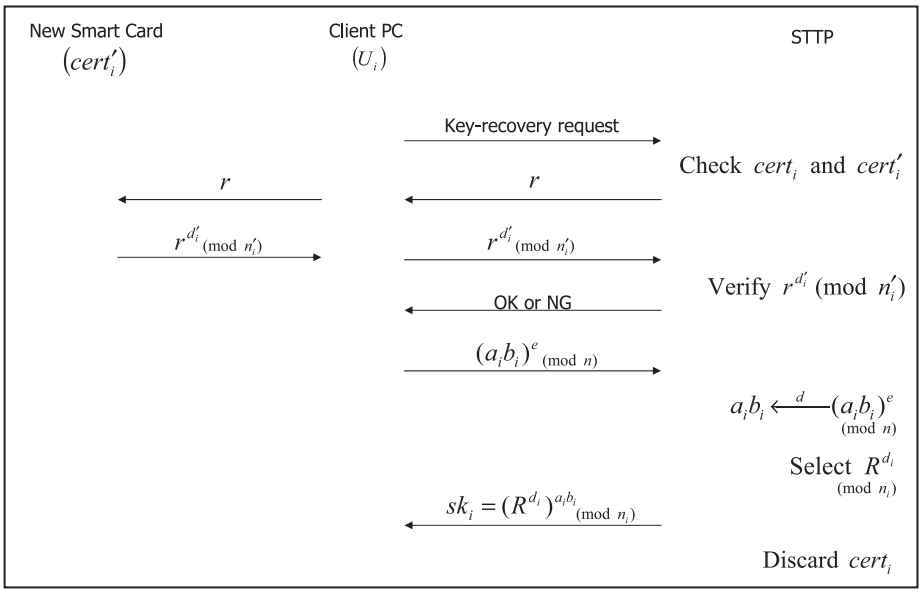

Fig. 5 Key-recovery phase.

STTP sends a random number r to U's new smart card through the client PC as a challenge. The new smart card calculates $r^{d_i'} \bmod n_i'$ and sends $r^{d_i'} \bmod n_i'$ to the STTP. The new secret key d_i' is related to $cert_i'$. The STTP checks $r^{d_i'} \bmod n_i'$ using $cert_i'$. If $r^{d_i'} \bmod n_i'$ is valid, the STTP returns the message of OK to the client PC. Otherwise, the STTP returns the message of NG. Then the client PC exposes $(a_i b_i)^e \bmod n$ to the STTP server, and the STTP server decrypts it to $a_i b_i$ by using her decryption key d. Then, the STTP selects the value of $R^{d_i} \bmod n_i$ corresponding to U_i. The STTP computes $(R^{d_i})^{a_i b_i} \bmod n_i$ and sends it to the client PC. After recovering the disk encryption key, you discard it and start from the registration phase (2) again. Finally, the STTP adds the previous certificate $cert_i$ to *CRL*.

5. Experiment

Our scheme uses a smart card with a low processing ability, so its processing time might be long because two kinds of phases (the local authentication phase and the key-generation phase) are executed every time the client PC is used. We therefore measured the processing time required for each function in both phases in order to confirm that the smart card can complete the processing within a reasonable time.

For a full disk encryption, it is necessary for a program to decrypt the disk data before booting up the OS. We use a Virtual Machine Monitor (VMM) as this program. The VMM encrypts/decrypts the disk data by using it own encryption engine by which the disk access data is compulsorily hooked. After the smart card is authenticated, the client PC acquires the disk encryption key and then boots up the OS on the VMM. The values of $(a_i b_i)^e \bmod n$ and $(R^{d_i a_i b_i}) \bmod n_i$ are saved to the non-encrypted storage that the VMM manages.

Table 1 Experimental results.

Measurement items	Time
1. Acquisition of user's public key certificate	2140 ms
2. Verification of user's public key certificate	19.4 ms
3. Generation of challenge	19.2 ms
4. Generation of blind signature	420 ms
5. Verification of blind signature	11.7 ms
6. Generation of disk encryption key	0.202 ms
Total time	2610 ms

We used as the client PC a ThinkPad X60 (CPU: Core 2 Duo 2 GHz, Memory: 1 GB), used as the smart card an eLWISE (NTT Communications), and used as the smart card reader an ASE drive IIIE (Athena Smartcard Solutions). This smart card is equipped with a CPU, RAM and ROM, and corresponds to PKCS#11. We used Linux Fedora Core 6 as a host OS and Windows XP as a guest OS. The software we used was a smart card library group, the encryption library OpenSSL 0.9.8 b, the multiple-precision arithmetic library GMP 4.1.4-9, and the virtual machine monitor QEMU 0.8.2.

In the local authentication phase and the key-generation phase, we measured the processing time for the six items listed in Table 1. The six items are processed in both phases. Items 1 and 4 were executed in the smart card, and the others were executed in the VMM. The measurement was conducted by inserting the gettimeofday function in the VMM. We assume that the timer of VMM is correct. Each of the times for the items listed in Table 1 is the average of five measurements. The total time is discussed in Sect. 6.

The processing times for acquiring the user's public key certificate and for generating the blind signature were both comparatively long, and that for acquiring the public key certificate was the longest. Additionally, the processing

time for generating the blind signature contains not only the calculation but also the transfer of the signature data. That is, the transfer between the smart card and the client PC took longer than the signature calculation in the smart card. The discussion related to the experimental results will be conducted in Sect. 6.

6. Discussion

6.1 Protection of Disk Data

We assume that a malicious STTP and a non-STTP adversary try to acquire U_i's disk encryption key sk_i.

Theorem 1: A malicious STTP cannot acquire sk_i before the key-recovery phase, unless she obtains both U_i's smart card and pw_i.

Proof: A malicious STTP knows $R^{a_i b_i} \bmod n_i$ and $R^{d_i} \bmod n_i$ in U_i's registration phase. If the STTP obtains U_i's smart card without pw_i, she can do nothing but conduct the dictionary attack against U_i's smart card owing to tamper resistant. However, it is difficult for a malicious STTP to get sk_i because of its lock function (see Sect. 4.3). Conversely, even if the STTP obtains pw_i without U_i's smart card, she cannot take advantage of pw_i to obtain sk_i because sk_i is independent of pw_i. Furthermore, the STTP cannot compute the disk encryption key $sk_i = R^{a_i b_i d_i} \bmod n_i$ from both $R^{a_i b_i} \bmod n_i$ and $R^{d_i} \bmod n_i$ by Assumption 1. Therefore, a malicious STTP cannot acquire sk_i. □

If the STTP cracks U_i's client PC or get both U_i's smart card and pw_i, she can obtain sk_i, although she cannot derive U_i's secret key d_i. When the STTP cracks U_i's client PC, she can get the values of $R^{a_i b_i} \bmod n_i$ and $(a_i b_i)^e \bmod n$. Note that only the STTP can know sk_i after the recovery phase.

Theorem 2: A non-STTP adversary cannot acquire sk_i unless she obtains all of U_i's smart card, pw_i and the cracking ability of U_i's client PC.

Proof: If a non-STTP adversary obtains U_i's smart card and the cracking ability of U_i's client PC, she can get $(a_i b_i)^e \bmod n$ from U_i's client PC. However, since a non-STTP adversary cannot decrypt $(a_i b_i)^e \bmod n$ to $a_i b_i$, she can eventually do nothing but conduct the dictionary attack against U_i's smart card owing to tamper resistant. Hence, it is difficult to get sk_i because of its lock function. If a non-STTP adversary obtains the cracking ability of U_i's client PC and pw_i, it is difficult for such an adversary to get sk_i for the same reason as the above-mentioned. Furthermore, if a non-STTP adversary obtains U_i's smart card and pw_i, she can access U_i's smart card. However, she cannot acquire $R^{a_i b_i d_i} \bmod n_i$, since she does not know $R^{a_i b_i} \bmod n_i$. Therefore, a non-STTP adversary cannot acquire sk_i. □

If a non-STTP adversary obtains all of U_i's smart card, pw_i and the cracking ability of U_i's client PC, she can obtain sk_i from both $R^{a_i b_i} \bmod n_i$ and U_i's smart card. A non-STTP adversary can get $R^{a_i b_i} \bmod n_i$ from U_i's client PC.

6.2 Authentication of Recovered Key

A user has to reissue or initialize his smart card before recovering the disk encryption key. Hence he conducts the user authentication using such new smart card before the user starts the key recovery. This means that if the user U_i does not have the new smart card then an adversary cannot launch the key-recovery phase by impersonating U_i.

The STTP knows the values of $R^{a_i b_i} \bmod n_i$ and $R^{d_i} \bmod n_i$ corresponding to user U_i in the registration phase. In key-recovery phase, she can verify $R^{a_i b_i d_i} \bmod n_i$ using both $R^{d_i} \bmod n_i$ and $a_i b_i$. The STTP obtains $(a_i b_i)^e \bmod n$ from U_i's client PC and then decrypts it to $a_i b_i$. Therefore, the recovered disk encryption key can be authenticated by the STTP.

6.3 Revocation of Disk Encryption Key

The STTP can revoke the RSA private key d_i by using the *CRL*. The disk encryption key includes user's private key d_i. Therefore the STTP can revoke U_i's disk encryption key by revoking d_i. As a result, the user whose disk encryption key is revoked cannot decrypt his disk data even with his smart card. Of course, he cannot also use the PKI authentication with the same card after revocation. Note that it is necessary to have the *CRL* stored in the client PC updated.

6.4 Time until the OS has Booted Up

The time until the OS has booted up is the total time required for the local authentication phase, the key-generation phase, and the OS booting. Our experimental results showed that execute both the local authentication phase and the key-generation phase took about 2.6 seconds. On the other hand, booting up the OS (Windows XP) on the QEMU took about 60 seconds on the same machine. So the processing time for both phases was less than 5% of the time required for booting up the OS. We therefore think that the time which is required for both phases is short enough to be practical.

6.5 Length of Disk Encryption Key

The standardization of full disk encryption is discussed in [18], [19]. In these documents, two kinds of standardization of narrow-block encryption and wide-block encryption are advanced at the same time. In these standardizations, some encryption modes are elected as a candidate. Among these modes the key length of XTS mode and TET mode is two block lengths and the key length of EME* mode is three block lengths. Hence the mechanism of our scheme is meaningful because it generates the disk encryption key of two or more block lengths.

7. Conclusion

We proposed a scheme which enables to recover the disk

encryption key when the user's smart card is lost. In our scheme, the disk encryption key is not preserved anywhere and then the STTP cannot know the key before key-recovery phase. Furthermore, we showed in experiments that it took about 2.6 seconds for the smart card to execute both the local authentication phase and the key-generation phase. Both phases are used every time the OS boots up. This time is a short time compared with the OS booting time. Therefore, we found that this processing time is short enough to be practical.

In our key-recovery phase, the disk encryption key is exposed to the STTP. Hence the security against semi-honest STTP seems weak. We would like to execute the key-recovery phase without exposing the disk encryption key to the STTP as a future work.

Acknowledgments

This work is supported by Special Coordination Funds for Promoting Science and Technology of Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] K. Omote and K. Kato, "Protection and recovery of disk encryption key using smart cards," Proc. 5th International Conference on Information Technology: New Generations – ITNG'08, IEEE, pp.106–111, 2008.
[2] N. Ferguson, "AES-CBC +Elephant diffuser: A disk encryption algorithm for windows vista," Microsoft Corp., 2006.
[3] J. Hughes and C. Feist, "Architecture of the secure file system," Proc. 18th IEEE Symposium on Mass Storage Systems and Technologies – MSS'01, IEEE, pp.277–290, 2001.
[4] W.C. Ku and S.M. Chen, "Weakness and improvements of an efficient password based remote user authentication scheme using smart cards," IEEE Trans. Consum. Electron., vol.50, no.1, pp.204–207, 2004.
[5] C.C. Chang and J.S. Lee, "A smart-card-based remote authentication scheme," Proc. 2nd International Conference on Embedded Software and Systems – ICESS'05, IEEE, pp.445–449, 2005.
[6] E.J. Yoon and K.Y. Yoo, "More efficient and secure remote user authentication scheme using smart cards," Proc. 11th International Conference on Parallel and Distributed Systems – ICPADS'05, pp.73–77, IEEE, 2005.
[7] M.S. Hwang and L.H. Li, "A new remote user authentication scheme using smart cards," IEEE Trans. Consum. Electron., vol.46, no.1, pp.28–30, 2000.
[8] H.M. Sun, "An efficient remote user authentication scheme using smart cards," IEEE Trans. Consum. Electron., vol.46, no.4, pp.958–961, 2000.
[9] Y. Wang and P. Dasgupta, "Remote user authentication using VMM-based security manager," http://cactus.eas.asu.edu/PARTHA/Papers-PDF/2006/authentication_yw.pdf, 2006.
[10] S. Lim, S. Kang, and J. Sohn, "Modeling of multiple agent based cryptographic key recovery protocol," Proc. 19th Annual Computer Security Applications Conference – ACSAC'03, pp.119–128, IEEE, 2003.
[11] M.J. Markowitz and R.S. Schlafly, "Key recovery in secretagent," Digital Signature, 1997.
[12] R. Gennaro, P. Karger, S. Matyas, M. Peyravian, A. Roginsky, D. Safford, M. Willett, and N. Zunic, "Secure key recovery," IBM Thomas J. Watson Research Center, 1999.
[13] K. Narimani and G.B. Agnew, "Key management and mutual authentication for multiple field records," Proc. 3rd International Conference on Information Technology: New Generations – ITNG'06, pp.568–569, IEEE, 2006.
[14] M. Blaze, "Key management in an encrypting file system," Proc. the USENIX Summer 1994 Technical Conference, pp.27–35, 1994.
[15] M. Kwon and Y. Cho, "Protecting secret keys with blind computation service based on Discrete logarithm," Proc. 18th International Workshop on Information Security Applications, pp.312–315, 2003.
[16] Wikipedia, "Full disk encryption," http://en.wikipedia.org/wiki/Full_disk_encryption, 2007.
[17] M. Manbo and H. Shizuya, "A note on the complexity of breaking Okamoto-Tanaka ID-based key exchange scheme," IEICE Trans. Fundamentals, vol.E82-A, no.1, pp.77–80, Jan. 1999.
[18] SISWG, "P1619: Standard architecture for encrypted shared storage media," IEEE Project 1619 (P1619), 2007.
[19] SISWG, "P1619.2: Standard for wide-block encryption for shared storage media," IEEE Project 1619.2 (P1619.2), 2006.

Kazumasa Omote received his M.S. and Ph.D. degrees in information science from Japan Advanced Institute of Science and Technology (JAIST) in 1999 and 2002, respectively. He joined Fujitsu Laboratories, LTD from 2002 to 2008 and engaged in research and development for network security. He has been a research assistant professor at the Japan Advanced Institute of Science and Technology (JAIST) since 2008. His research interests include applied cryptography and network security. He is a member of the IPS of Japan.

Kazuhiko Kato received the BE and ME degrees from the University of Tsukuba, Japan, in 1985 and 1987, respectively. He received the PhD degree from the University of Tokyo, Japan, in 1992. From 1989 to 1993, he was a research associate in the Department of Information Sciences, Faculty of Sciences at the University of Tokyo. He is currently a professor in the Department of Computer Science, Graduate School of System Information Engineering at the University of Tsukuba. His research interests include operating systems, distributed systems, and secure computing. He received the distinguished paper awards from JSSST and IPSJ in 2004 and 2005, respectively.