Fifth International Conference on Information Technology: New Generations

Protection and Recovery of Disk Encryption Key using Smart Cards

Kazumasa Omote and Kazuhiko Kato
Department of Computer Science, Graduate School of ISE,
University of Tsukuba

Abstract

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 encryption key protected securely. Disk security has been improved by storing the encryption key in a hardware token such as a smart card or USB device. There must be some way to recover the encryption key when the token is lost, but to prevent information leakage the encryption key should not be known by the system administrator and should not be able to be recovered by malicious users inside the system.

Here we describe a scheme that can limit key recovery when the user's smart card is lost and can do so without the administrator knowing the key. The smart card is used for generating the key and for improving the user authentication.

1. Introduction

Information leakage has recently become a serious problem and most often a result of actions inside rather than outside the system that should be protecting the information. 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.

Information leakage from a disk in a managed network (e.g., an enterprise network) is generally prevented by encrypting either the entire disk or just the confidential files stored on it. Since this encryption uses highly secure symmetric-key encryption algorithms, however, it is not easy for a user to memorize the random numbers constituting the disk encryption key. So it is important not only to encrypt the disk data but also to store the disk encryption key securely. This can be done by storing the key in a hardware token such as smart card or USB device, but there must also be some way to recover the it if the token is lost. For example, it is necessary to keep a backup copy in a safe place such as another key management server.

The encryption key should not be known by the system administrator, however, nor should it be possible for malicious users within the system to recover a user's encryption key.

In this paper we present a scheme that can limit key recovery when a user loses his smart card and can do so without the administrator knowing the key. In our scheme the disk encryption key is not preserved anywhere and only someone who has a user's smart card and knows the user's password can decrypt that user's disk data. The smart card is used for generating the key and for improving the user authentication.

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

2. Related work

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 [1].

The BitLocker feature of the Windows Vista OS is a popular tool 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], [5], [6], [7], [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], [11], [12], [13], [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 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 administrator does not know the key in advance. This means that the encryption key can be recovered by a malicious user. Although we can easily devise a method that uses the key with a certificate, in that case the key would be known to the administrator in advance. Key-

978-0-7695-3099-4/08 $25.00 © 2008 IEEE
DOI 10.1109/ITNG.2008.195

recovery methods using a smart card have been proposed [13], [14], but they are fundamentally different from the one in our scheme.

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. Basic policy

In this section we describe basic policy for constructing our scheme. We assume that the scheme is used in managed network such as an enterprise network.

3.1. Realization of full disk encryption

An advantage of full disk encryption (encrypting all byte data on the disk) 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 [1]. 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.

3.2. Use of a Virtual Machine Monitor

For a full disk encryption, it is necessary for a program to decrypt the disk data before starting 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. The VMM first authenticates the smart card locally by using the public key authentication method. When the card is authenticated, the VMM can acquire the disk encryption key and load it into the VMM memory. Note that we do not put secret information such as the disk encryption key directly into the VMM. The validity of the signature calculation in the smart card can be guaranteed at the same time that the card is being authenticated.

3.3. Use of smart card

The smart card is used for generating the key and for improving the authentication. Because we use a symmetric-key encryption algorithm, it is difficult for a user to memorize the disk encryption key (e.g., a 128-bit random number). Our scheme therefore generates the encryption key with the help of a smart card. We assume that the smart card is also an identification card (ID card), so the 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. 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.

3.4. Use of Semi-Trusted Third Parties

The administrator and the trusted management server together constitute a semi-trusted third party (STTP) that issues legitimate identification and smart cards, recovers lost encryption keys, and prevents the information stored in its database from being falsified. It need not, however, prevent the leakage of a user's secret information, such as the encryption key, because it does not have any of the user's confidential information. The only secret information that the STTP has is its own private key.

3.5. Recovery of disk encryption key

If you have lost your smart card, you might not be able to decrypt your disk data because the disk encryption key is stored in your smart card. We therefore need to have some way to recover the key when the smart card is lost. One drawback of most key recovery methods, however, is that they require the STTP to know the user's key. Although there are also some methods in which the user's key need not be known to the STTP, with them it is difficult to ensure that a key can be recovered only by the legitimate user. This means that the encryption key can be recovered by a malicious user. We therefore want to recover the key without letting the STTP know what the key is.

4. Our scheme

In this section we explain the detailed protocol of a scheme following the basic policy described in Section 3.

4.1. Premises

The premises of our scheme are these:

1. The STTP issues the public key certificate and the certificate revocation list, and it prevents this information from being falsified while it is in the STTP's database.
2. 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.

3. The calculation algorithms (e.g., signature generation) in the smart card are not falsified.
4. 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.
5. The user can read data other than his private key in his smart card by using his password, but he cannot alter it.
6. The code of the VMM in the client PC is not falsified, and the STTP does not steal the user's private information in the VMM.
7. We do not deal with the situation in which the user who has lost his smart card forgets his password.

4.2. 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, r_i	:	Random numbers ($\leq$| n |)
a_i, b_i	:	Random numbers ($\leq$| 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
$h(\cdot)$	:	Hash function (e.g. SHA1)

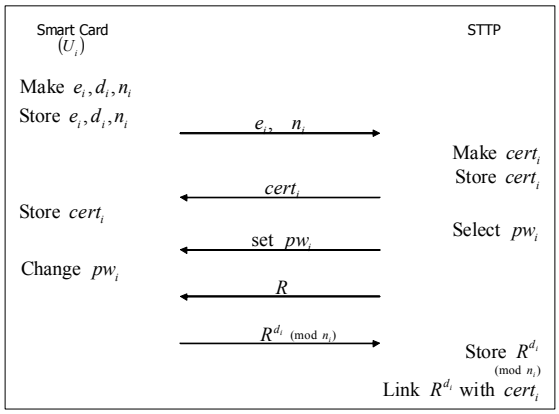

Figure 1: Registration phase (1) (Smart Card -- STTP)

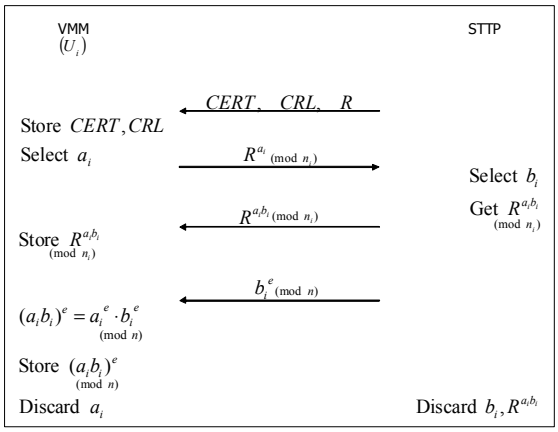

Figure 2: Registration phase (2) (VMM -- STTP)

4.3. Protocol

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

A. Initial phase
The STTP generates its RSA keys, *CERT*, and *CRL*. It also generates the random numbers R that are the common public information of the system.

B. Registration phase (1)
The user connects his smart card with the STTP and initializes the card (see Figure 1). In this part of the registration phase the user U_i sets his RSA keys, $cert_i$, and pw_i in his smart card. The STTP makes each user's public information $R^{d_i} \pmod{n_i}$ by using U_i's smart card and preserves the value along with $cert_i$.

C. Registration phase (2)
U_i connects the client PC (VMM) to the STTP with a secure channel, and makes information that is necessary for the key recovery in the VMM (see Figure 2). In this part of the registration phase the VMM generates both $R^{a_i b_i} \pmod{n_i}$ and $(a_i b_i)^e \pmod{n}$ in cooperation with the STTP. Note that neither the VMM nor the STTP knows the value of $a_i b_i$.

D. Local authentication phase
The client PC (VMM) confirms the validity of the smart card in a local network. We can confirm that the smart card was legitimately issued by the STTP. In this phase we use a blind signature for both the generation of the disk encryption key and the authentication. The procedure is shown in Figure 3.

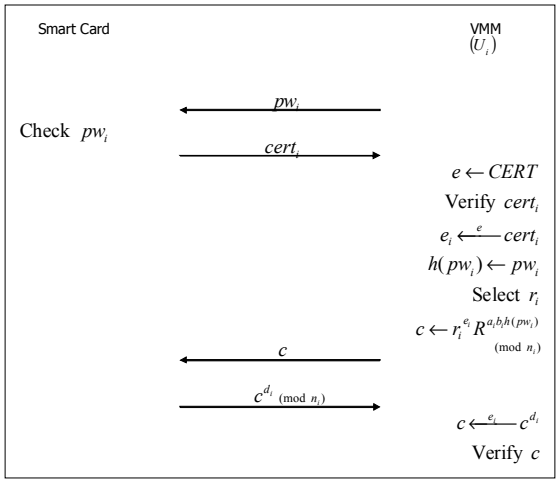

Figure 3: Local-authentication phase

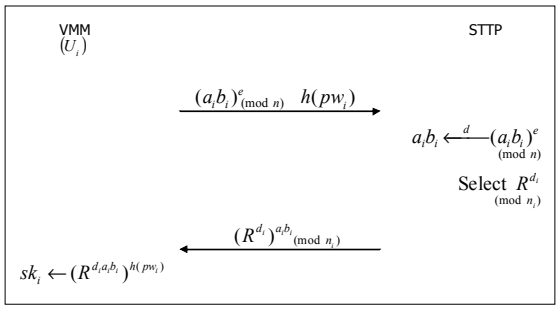

Figure 4: Key-recovery phase

E. Disk encryption key generation phase

When a local authentication succeeds, the VMM executes the following procedure.

1. The VMM derives the legitimate disk encryption key sk_i of size $|n_i|$ by calculating $c^{d_i}/r_i \pmod{n_i}$:

$$sk_i = R^{a_i b_i h(pw_i) d_i} \pmod{n_i}.$$

2. The VMM 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. For instance, when the symmetric-key encryption algorithm is AES-128, you can get eight disk encryption keys because $|n_i| = 1024$ bits. In this case you use only the first key, sk_{i1}.

$$sk_i = sk_{i1} \,|\, sk_{i2} \,|\cdots$$

F. Key-recovery phase

When you accidentally lose your smart card, you execute the key-recovery phase by connecting your client PC (VMM) to the STTP with a secure channel (see Figure 4). After recovering the key, you discard it and start from registration phase (2) again.

5. Experiment

5.1. Purpose

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 disk encryption 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.

5.2. Circumstances

We used as the client PC a ThinkPad X60 (CPU: Core 2 Duo 2GHz, Memory: 1GB), 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.

The software we used was the OS Linux Fedora Core 6, a smart card library group, the encryption library OpenSSL 0.9.8b, the multiple-precision arithmetic library GMP 4.1.4-9, and the virtual machine monitor QEMU 0.8.2.

5.3. Method and Results

In the local authentication phase and the disk encryption key generation phase, we measured the processing time for the six items listed in Table 1. 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 source code.

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

Each of the times for the items listed in Table 1 is the average of five measurements. The total time is discussed in subsection 6.5.

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.

6. Discussion

6.1. Limitation of recovered key

The value of $R^{d_i} \pmod{n_i}$ can be calculated only from U_i's smart card because that is the only place where the private key d_i is stored. That is, the value of $R^{d_i} \pmod{n_i}$ is link to U_i through $cert_i$ (see Figure 1). And the recovered disk encryption key $R^{a_i b_i h(pw_i) d_i} \pmod{n_i}$ is calculated from the $R^{d_i} \pmod{n_i}$ that the STTP stores (see Figure 4). That is, the recovered encryption key is link to the value of $R^{d_i} \pmod{n_i}$. The recovered encryption key is therefore linked to U_i. This means that the STTP can limit U_i's recovered encryption key to the value of $R^{a_i b_i h(pw_i) d_i} \pmod{n_i}$ if the STTP authenticates U_i who lost his smart card.

6.2. Impersonation attack by STTP

The RSA private key d_i, the hash value of the password $h(pw_i)$, and the value of $a_i b_i$ —all corresponding to U_i —are needed to generate the disk encryption key, but neither the STTP nor an attacker knows these values. The STTP does not maintain any of the user's secret information.

The STTP can get the values of $R^{a_i b_i} \pmod{n_i}$ and $R^{d_i} \pmod{n_i}$ in the registration phase but, because of the secrecy of the RSA cryptosystem, it cannot get $a_i b_i$ or d_i from these values. Therefore neither the STTP nor an attacker impersonating the STTP can get the user's disk encryption key

U_i cannot get $a_i b_i$ or d_i. So the user cannot get the disk encryption key without using his smart card.

6.3. Revocation of disk encryption key

When the disk encryption key of some user is revoked, the user must not be able to freely use his 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 encryption key is revoked cannot decrypt his disk data even with his smart card. Of course he is also unable to use the PKI authentication with the same card after revocation. Note that it is necessary to have the *CRL* stored in the VMM updated.

6.4. Protection of recovered key

The disk encryption key should not be known by anybody even after it is recovered. In our scheme the STTP cannot derive U_i's encryption key after it is revoked because the STTP does not know the hash value $h(pw_i)$. Also, the person other than STTP does not know $h(pw_i)$ by the same token. The encryption key is thus kept secret even after the key recovery phase.

6.5. 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 disk encryption key generation phase, and the OS booting. Our experimental results showed that execute both the local authentication phase and the disk encryption 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.

6.6. Length of disk encryption key

The standardization of full disk encryption is discussed in [16], [17]. 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. Summary

In this paper we described a scheme that can limit the recovered encryption key without informing the STTP (effectively, the system administrator) of the user's key. Although the STTP maintains none of the user's confidential information, the user can recover the encryption key by cooperating with the STTP. Moreover, we showed in experiments that it took about 2.6 seconds

for the smart card to execute both the local authentication phase and the disk encryption key generation phase. This is a short time compared with the OS booting time.

Acknowledgements

This work was supported by Special Coordination Funds for Promoting Science and Technology that were provided by the Japanese Ministry of Education, Culture, Sports, Science and Technology. We are also grateful to the Secure VM project members who advised us and cooperation in this research.

References

[1] Wikipedia, "Full disk encryption,"http://en.wikipedia.org/wiki/Full_disk_encryption, 2007.

[2] Niels Ferguson, "AES-CBC +Elephant diffuser: A Disk Encryption Algorithm for Windows Vista," Microsoft Corp., 2006.

[3] J. P. Hughes and C. J. Feist, "Architecture of the Secure File System," Proceedings of the Eighteenth IEEE Symposium on Mass Storage Systems and Technologies (MSS'01), 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. on Consumer Electronics, 50(1), 2004.

[5] C. C. Chang and J. S. Lee, "A Smart-Card-Based Remote Authentication Scheme," Proceedings of the Second International Conference on Embedded Software and Systems (ICESS'05), 2005.

[6] E. J. Yoon and K. Y. Yoo, "More Efficient and Secure Remote User Authentication Scheme Using Smart Cards," Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05), 2005.

[7] M. S. Hwang and L. H. Li, "A New Remote User Authentication Scheme Using Smart Cards," IEEE Trans. on Consumer Electronics, 46(1), 2000.

[8] H. M. Sun, "An Efficient Remote User Authentication Scheme Using Smart Cards," IEEE Trans. on Consumer Electronics, 46(4), 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," Proceedings of the 19th Annual Computer Security Applications Conference (ASAC 2003), 2003.

[11] M. J. Markowitz and R. S. Schlafly, "Key Recovery in SecretAgent," Digital Signature, 1997.

[12] R. Gennaro et al., "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," Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06), 2006.

[14] M. Blaze, "Key Management in an Encrypting File System," Proceedings of the USENIX Summer 1994 Technical Conference, 1994.

[15] M. Kwon and Y. Cho, "Protecting Secret Keys with Blind Computation Service," Third International Workshop on Information Security Applications, 2002.

[16] SISWG, "P1619: Standard Architecture for Encrypted Shared Storage Media," IEEE Project 1619 (P1619), 2007.

[17] SISWG, "P1619.2: Standard for Wide-Block Encryption for Shared Storage Media," IEEE Project 1619.2 (P1619.2), 2006.