A Key Generation Scheme of Self-Encryption based Mobile Distributed Storage System

Yoshihiro Kawahara, Hiroki Endo and Tohru Asami

The University of Tokyo, 7-3-1 Hongo Bunkyo-ku Tokyo 113-8656, Japan

Abstract. Mobile phones are frequently lost or stolen. Latest mobile handsets contain important information such as an address book, short mail messages, and e-cash. To prevent a stranger from accessing to such private information, practical security mechanisms have to be introduced into mobile handsets. We have developed a distributed network storage system that protects private data files stored on the mobile handsets without demanding complex operations for users. As computation resource on the mobile handsets is limited, a lighter encryption scheme is indispensable. In this paper, we report a self-encryption scheme for mobile distributed storage system. Different from existing encryption schemes such as PKI (Public Key Infrastructure), this scheme exploits a diversity of data files for generating unique encryption keys while minimizing computation overhead of encryption. Experimental results show that our scheme can generate completely random keys from zipped text files with light operations.

1 Introduction

Advanced mobile handset features such as digital camera, music player, and e-cash have made mobile phones so convenient and attractive for all people. The most recent mobile handsets are equipped with high-speed wireless link and gigabytes of storage, and multiple CPUs for different purposes. Its performance is far better than the computers of several decades ago. Along with such technological advances, each mobile handset came to contain various kinds of important information such as an address book, short mail messages, and e-cash. However, mobile handsets are so small that they are easily lost or stolen. Tokyo metropolitan police department reports that more than 100,000 mobile handsets have been lost or stolen only in Tokyo area in 2006 [1]. Currently, mobile operators provide a networked based blocking service that blacklists the user's phone so it will be unusable until picked up by the owner. However, such a service is helpless for preventing a stranger from accessing to the private data stored inside the mobile handsets.

In this paper, we first review technical requirements for security mechanisms of mobile handsets in preparation for lost and theft. In general, there are design tradeoffs between ease of use and safety. As the computational and communication resources of mobile handsets are limited, general security mechanisms are not always

Kawahara Y., Endo H. and Asami T. (2007).
A Key Generation Scheme of Self-Encryption based Mobile Distributed Storage System.
In *Proceedings of the 1st International Joint Workshop on Wireless Ubiquitous Computing*, pages 3-12
DOI: 10.5220/0002425200030012
Copyright © SciTePress

applicable. Therefore we clarify design criteria peculiar to mobile devices in order to maximize the usability while minimizing security risks. Then we describe a design of mobile distributed storage system based on self-encryption scheme. The self-encryption scheme refers to an encryption scheme which we define in this paper. An encryption key management scheme is one of the main challenges in the security community. Conventional block ciphers aim at generating stronger cipher text from plain text while minimizing the length of the encryption key. On the other hand, our self-encryption scheme exploits the heterogeneity of data files (plain text) to generate a longer encryption key sequence. As we use a longer key, we can use a simpler stream cipher which requires less computational resources. By integrating the self-encryption scheme into distributed storage system, we aim at realizing a practical secure file system for mobile handsets.

2 Security Technologies for Mobile Handsets

In this section, we introduce security technologies provided for mobile communication services such as mobile phones and PHS (Personal Handy-phone System). Then we clarify the technical requirements and design criteria for data protection of mobile handsets.

2.1 Existing Approaches

Existing security mechanisms for mobile handsets include unauthorized access prevention mechanisms, data backup mechanism, and data encryption mechanisms for locally stored data. Table 1 shows security mechanisms provided for mobile handsets such as laptop PC and mobile phones. When a user's mobile terminal is lost or stolen, terminal security mechanisms can reduce the risk of strangers to access to confidential data stored in the terminal. However, if the malicious attacker breaks the chassis to pick out HDD or a flash memory, data may be analyzed easily. When data security mechanisms are applied to the local storage data, the data can be protected with cryptographic assurance.

If we look at existing data security services for mobile handsets, a remote deletion service is the only feasible choice. With remote lock services, the owner can lock the mobile handset and/or delete locally stored data by sending a special command over wireless network [2,3]. This approach, however, is helpless if the mobile handset is intentionally placed outside the coverage area. Though data encryption and remote storage can be the potential choices for sustaining data security, actual realization mechanisms are sometimes too heavy for resource constrained hardware like mobile handsets[4,5]. Remote storage approach works without a local data storage and downloads necessary files from a remote storage over the network. This approach is acceptable for desktop PC and workstations that are always connected to high speed LAN. Mobile handsets, on the other hand, are connected to the network by wireless link, which is, in general, error-prone and limited in bandwidth. Therefore downloading the entire file may lose usability. A distributed storage divides a file into several pieces and stores them in different places in order to increase data confidentiality and availability at the same time.

Shamir's secret sharing scheme [6] can be used as a method to realize a distributed storage. This scheme divides data D into n pieces in such a way that D is easily reconstructable from any k pieces, but even complete knowledge of k - 1 pieces reveals absolutely no information about D. This technique enables the construction of robust key management schemes and cryptographic schemes that can function securely and reliably. As this scheme is based on polynomial interpolation, even the most efficient algorithms are too heavy to implement as a file system on the resource limited mobile handsets.

Table 1. Security Mechanisms for Mobile Terminals.

		Laptop PC	Mobile Phones
Terminal Security	Biometric Authentication	✓	✓ (finger print)
	Secure IC Memory	✓ (SD, Felica, etc.)	✓ (SIM)
	BIOS Password	✓	NA
	Log-in Password	✓	✓ (PIN code)
Data Security	Authentication / Authorization	✓	NA
	Encryption	✓	NA
	Remote Lock	✓	✓
	Remote Storage	✓	NA

2.2 Capabilities of Mobile Handsets

Computation and communication resources of mobile handsets such as PDA and mobile phones are limited compared to personal computers and workstations. Some of the mobile handsets are equipped with wireless LAN (IEEE 802.11) and Bluetooth interface for local area communication. As a global connection, 3G and GPRS data connection services are available. In Japanese market, HSDPA (High Speed Downlink Packet Access) and CDMA2000 1xEV-DO services are provided as 3G service and those services offer upto several Mbps communication[7].

Other than the difference of abilities, mobile handsets afford less complex operations for users because its user interfaces is constrained (a tiny screen and several pieces of keys) and a range of potential users are so wide regardless of age and sex.

2.3 System Design

Fig. 1 shows a basic design of our system. In the mobile handset, confidential data is encrypted and divided into a pair of cipher text files by using a self encryption scheme (which will be described in next section). Each piece of cipher text cannot be decrypted alone. One of the cipher text files is stored in the local storage area of the mobile handset and the other piece is automatically uploaded to remote storage. Moreover, the piece stored locally can be copied to another remote storage for backup purpose. When a user hopes to use a file, the file system of the mobile handset seeks for the local and remote pieces and decrypts the cipher text files. When the mobile handset is lost or stolen, the user blocks the access to remote storage to prevent decryption of confidential files. When the user hopes to switch to a new mobile handset, he just needs to copy the backed up pieces to the new handset.

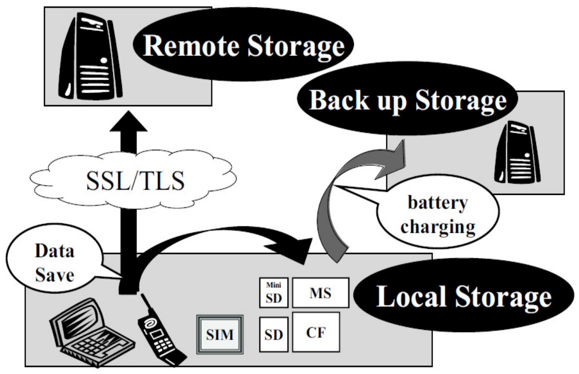

Fig. 1. A System Design. A secret data file is divided into two pieces of plaintexts. One of them is scrambled and used as a key. The key is uploaded to a remote server. The other piece is encrypted by the key and stored locally. The locally stored ciphertext is sometimes backed up to a back up storage.

As this scheme needs to download a piece of a file from the remote storage and decrypt it, the user may experience short delay in using files. However, most text-based files can be downloaded within several seconds when we use high speed wireless network such as HSDPA. Multimedia data such as MP3 files requires longer time for downloading.

3 Self Encryption Scheme

In this section, we present a self encryption scheme for mobile distributed storage system. In this paper, we define self encryption scheme as an encryption scheme whose encryption key is generated from the information contained in the target file itself. In other words, self encryption scheme exploits the diversity inherent in data files, and uses various information such as meta-data and a part of plain text as an input of key generation process. As a self-encryption scheme automatically generates unique keys, users do not need to worry about key management.

3.1 Block Cipher and Stream Cipher

In general, typical encryption mechanism such as DES[8] and AES[9] are called a block cipher. A block cipher is a symmetric key cipher which operates on fixed-length groups of bits, termed blocks, with an unvarying transformation. When encrypting, a block cipher might take a (for example) 128-bit block of plaintext as input, and output a corresponding 128-bit block of ciphertext. The exact transformation is controlled using a second input — the secret key. Decryption is similar: the decryption algorithm takes, in this example, a 128-bit block of ciphertext together with the secret key, and yields the original 128-bit block of plaintext. To encrypt messages longer than the block size (128 bits in the above example), a mode of operation is used. Block ciphers can be contrasted with stream ciphers; a stream cipher operates on individual digits one at a time and the transformation varies during the encryption.

Stronger block cipher requires much computation overhead. A stream cipher, on the other hand, typically executes at a higher speed than block ciphers and has lower hardware complexity[10]. However, stream ciphers can be susceptible to serious security problems if the encryption key is generated from weak random sequence generators. In this sense, stream cipher requires stronger random sequence generator, which is typically provided as a hardware module.

Our approach, self encryption scheme, can generate a long key (random sequence) from the data file (plaintext). Therefore a plaintext can be encrypted at higher speed without any special hardware support. Moreover, the encryption key is unique to each plaintext, thus the entire system remain safe even if one of the ciphertext has been defeated by an attacker.

3.2 Formulation of Self Encryption Scheme

The whole process of self encryption scheme is described in Table 2, Figure 2 and 3. The encryption procedure is performed as follows:

1. A file content message M is split into M_L and M_R by a split function S
2. Generate an encryption key K_L from M_R by a function F_R
3. Encrypt(E_L) the message M_L using K_L to generate a ciphertext C_L
4. Store C_L in the local storage area
5. Generate a key K_R using C_L with a function F_L
6. Encrypt(E_R) M_R using K_R to generate a ciphertext C_R
7. Upload C_R to a remote storage area
8. Backup C_R to a backup storage

Then the decryption procedure is performed as follows:

1. Download C_R from the remote storage
2. Generate the key K_R from locally stored data C_L by F_L
3. Decrypt C_R using K_R to recover the message M_R
4. Generate the key K_L from M_R by F_R
5. Recover M_L by decrypting(E_L) C_L using K_L
6. Concatenate(S^{-1}) M_L and M_R to recover the original M

Table 2. Formulation of Self Encryption Scheme.

Encryption	
$\{M_L, M_R\} = S(M)$	Split M into M_L and M_R
$K_L = F_R(M_R)$	Generate K_L from M_R
$C_L = E_L(M_L)$	Encrypt M_L using K_L
$K_R = F_L(C_L)$	Generate K_R from C_L
$C_R = E_R(M_R)$	Encrypt M_R using K_R
Decryption	
$K_R = F_L(C_L)$	Generate K_R from C_L
$M_R = D_R(C_R)$	Decrypt C_R from K_R
$K_L = F_R(M_R)$	Generate K_L from M_R
$M_L = D_L(C_L)$	Decrypt C_L using K_L
$M = S^{-1}\{M_L, M_R\}$	Concatenate M_L and M_R into M

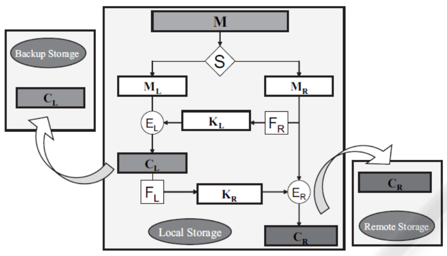

Fig. 2. Encryption Procedure.

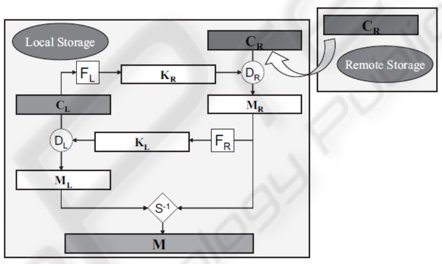

Fig. 3. Decryption Process.

3.3 Algorithms for Self Encryption

We presented a formulation of self encryption scheme in 3.1. When thinking of a real implementation of the self encryption scheme, we need to be careful in choosing algorithms and parameters. In this section, we mention the file split method S, the encryption algorithm E, and the key generation function F.

The main objective of splitting a message M into M_R and M_L is to generate a key for encryption. When using shorter M_R, it means that longer messages are encrypted with shorter keys. It makes, in general, easier to defeat the resulting ciphertext but requires shorter time for uploading (and downloading) the key from the remote server. If longer M_R is chosen, ciphertext becomes stronger. But note that the maximum length of M_R is 1/2 of the length of original message M. When the length of M_R and M_L is equal and the resulting keys K_R and K_L are random enough, ciphertext C_R and C_L become unbreakable.

In our implementation, XOR is used as an encryption algorithm E (E_R and E_L) to combine the plaintext with the key. This stream cipher based approach executes at a higher speed than block ciphers (such as AES and Triple DES) and have lower hardware complexity, which is very important in mobile handsets.

When using stream ciphers, we need to be careful in the key used for encryption. For stream ciphers to be safe, the key needs to be completely random and must not be used again. When M_R is shorter than M_L, we need to generate a longer key (K_L) from shorter message M_R by using a KDF (Key Derivation Function) based on hash functions such as SHA-1 and SHA-256[9]. When $|M_R| = |M_L|$, we can choose lighter shuffling algorithms.

4 Key Generation Functions

In this section, we look into the key generation functions F_R and F_L. We evaluate the strength of K_R and K_L which are generated by two different algorithms: A KDF using SHA-1 ($|M_R| < |M_L|$) and a proposed scrambler ($|M_R| = |M_L|$).

4.1 Methodologies

Figure 4 shows a key derivation function used in ANSI X9.63[12]. The message M is split into j pieces and fed to SHA-1 hash function along with counter values. By concatenating resulting stings $k_1 \ldots k_j$, the key K is achieved.

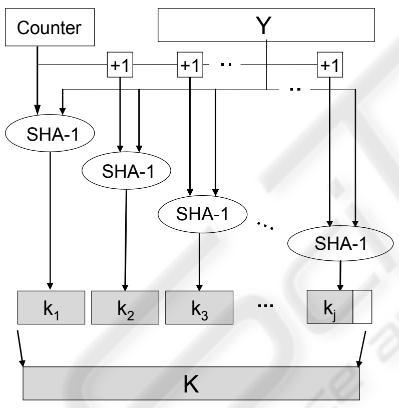

Fig. 4. A Key Derivation Function.

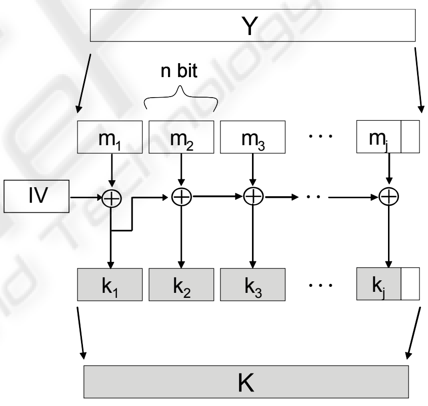

Fig. 5. Scrambling Algorithm.

Figure 5 shows our proposed scrambling algorithm. This algorithm splits input message Y with every fixed length (64, 128, 256, 512 bytes) to generate $m_1, m_2, \ldots, m_j$. Then calculate $k_1 = IV \oplus m_1$, $k_i = k_{i-1} \oplus m_i$. Here $\oplus$ denotes XOR operation and IV denotes initial value. We get the entire key K by concatenating $m_1, m_2, \ldots, m_j$.

4.2 Evaluation

We evaluate the strength of the key generated by the algorithms mentioned previously. The strength of the key is assessed as a randomness of symbols appeared in K. Various statistical tests can be applied to a sequence to attempt to compare and evaluate the sequence to a truly random sequence. A practical randomness assessment of a string is defined in NIST SP800-22[13]. A statistical test is formulated to test a specific null hypothesis (H_0). The null hypothesis under test is that the sequence being tested is random. Associated with this null hypothesis is the alternative hypothesis (H_a) which is that the sequence is not random. For each applied test, a decision or conclusion is derived that accepts or rejects the null hypothesis. Each test is based on a calculated test statistic value, which is a function of the data. The test statistic is used to calculate a P-value that summarizes the strength of the evidence against the null hypothesis. For these tests, each P-value is the probability that a perfect random number generator would have produced a sequence less random than the sequence that was tested, given the kind of non-randomness assessed by the test. If a P-value for a test is determined to be equal to 1, then the sequence appears to have perfect randomness. A P-value of zero indicates that the sequence appears to be completely non-random. A P-value ≥ 0.01 would mean that the sequence would be considered to be random with a confidence of 99 %. A P-value < 0.01 would mean that the conclusion was that the sequence is non-random with a confidence of 99 %. For practical reasons, 0.01 is often used as a threshold to test the sequence. In this paper, we use proportion and uniformity of p-value is assessed.

Table. 3. Proportion of p-value > 0.01.

Method	Material	Frequency	Runs	Rank	Cusum
(Original M)	Plain text Zip	0 (NG) 0.021 (NG)	0 (NG) 0.029 (NG)	0 (NG) 0.014 (NG)	0 (NG) **0.988 (OK)**
KDF	**Plain text** **Zip**	**0.992 (OK)** **0.991 (OK)**	**0.994 (OK)** **0.988 (OK)**	**0.994 (OK)** **0.993 (OK)**	**0.987 (OK)** **0.988 (OK)**
Scrambler (64 bytes)	Plain text **Zip**	0.540 (NG) **0.992 (OK)**	0.436 (NG) **0.987 (OK)**	0.514 (NG) **0.993 (OK)**	0.713 (NG) **0.985 (OK)**
Scrambler (128 bytes)	Plain text **Zip**	0.565 (NG) **0.988 (OK)**	0.567 (NG) **0.991 (OK)**	0.539 (NG) **0.987 (OK)**	0.992 (NG) **0.960 (OK)**
Scrambler (256 bytes)	Plain text **Zip**	0.489 (NG) **0.995 (OK)**	0.578 (NG) **0.983 (OK)**	0.383 (NG) **0.988 (OK)**	**0.960 (OK)** **0.984 (OK)**

Table 4. Uniformity of p-value.

Method	Material	Frequency	Runs	Rank	Cusum
(Original M)	Plain text Zip	0 (NG) 0 (NG)	0 (NG) 0 (NG)	0 (NG) 0 (NG)	0 (NG) **3.2×10^{-4} (OK)**
KDF	Plain text Zip	0 (NG) **0.244 (OK)**	7.3×10^{-8} (NG) **3.2×10^{-1} (OK)**	0 (NG) **0.123 (OK)**	1.3×10^{-6} (NG) 2.9×10^{-6} (NG)
Scrambler (64 bytes)	Plain text **Zip**	0 (NG) **0.850 (OK)**	0 (NG) **0.132 (OK)**	0 (NG) **0.738 (OK)**	0 (NG) **5.4×10^{-2} (OK)**
Scrambler (128 bytes)	Plain text **Zip**	0 (NG) **0.083 (OK)**	0 (NG) **0.138 (OK)**	0 (NG) **0.010 (OK)**	6.1×10^{-13} (NG) **1.2×10^{-3} (OK)**
Scrambler (256 bytes)	Plain text Zip	0 (NG) **0.070 (OK)**	0 (NG) **3.5×10^{-3} (OK)**	0 (NG) 0 (NG)	0 (NG) **1.4×10^{-2} (OK)**

We choose 1,000 documents available at copyright free online library named Aozora Bunko[14]. We choose cumulative sums (cusum) test, runs test, rank test, and frequency test as a measure. The result is shown in Table 3 & 4.

The experiment results show that random keys can be generated from zipped text files by using either KDF or proposed scrambler. On the other hand, it turned out that original plain text data is not suitable as source of keys. This is mainly due to the biased appearance probability of ascii code in the plain text data. As a zipped file consists of a sequence of randomized symbols due to the compression operation, randomness of plaintext is higher than uncompressed text message. In this sense, compressed files such as zipped file, JPEG, PNG picture files, MP3, MPEG multimedia files can be used as a good input data for generating a strong key. Plain text messages have to be compressed or randomized before key generation. As a randomize mechanism, it will appropriate to apply XOR with a random sequence generated by fast random sequence generators such as Mersenne Twister[15].

5 Conclusions

In this paper, we presented a self-encryption scheme for mobile distributed storage system. Self-encryption scheme exploits the heterogeneity of data files to automatically generate a longer encryption key sequence. A distributed storage system based on this approach has several advantages as follows:

1. As an encryption key is automatically generated from each data file, the keys become unique to the files. It means the key becomes stronger against attacks. Moreover, even if one of the keys is broken, the others stay safe.
2. Each encryption key can be generated by different ways. Users can optimize the cost and benefit by choosing the algorithms, the size of key (M_R and C_R), the encryption algorithm (E_R and E_L) depending on the available computation and communication resources and necessary security levels.
3. As a long and strong key can be generated, lighter encryption algorithm such as XOR based stream cipher can be chosen.

As an evaluation of our approach, we presented a randomness test of encryption keys generated in different ways. The evaluation results show that resulting keys are random as long as zipped files are fed as input source.

Currently, we are implementing our system into mobile handsets and see the feasibilities of our approach. The primal goal is to realize a user-friendly security mechanism that balances usability and security. In this sense, we will look for an integrated mechanism with existing security infrastructure such as PKI.

References

1. Tokyo Metropolitan Police Department: Annual report on lost and found service in 2006 (2007)
2. Remote Data Deletion Service, KDDI, http://www.kddi.com/business/pr/security/address/index.html
3. Omakase Lock Service, NTT DoCoMo http://www.nttdocomo.co.jp/service/anshin/lock

4. Sobti, S., Garg, N., Zheng, F., Lai, J., Shao, Y., Zhang, C., Ziskind, E., Krishnamurthy, A. and Wang, R.,: Segank: A Distributed Mobile Storage System, In Proceedings of 3rd Conference on File and Storage Technologies (FAST), (2004)
5. Kistler, J. J., and Satyanarayanan, M.: Disconnected operation in the Coda file system, In proceeding of the thirteenth ACM symposium on Operating systems principles, pp. 213-225. (1991)
6. Adi Shamir: How to share a secret, In Communications of the ACM 22, 1979, S. 612–613 (1979)
7. Dahlman, E.B. and Yu-Chuen Jou, Further evolution of 3G radio access, Communications Magazine, IEEE, Vol. 44, Issue 3, March 2006, pp. 34 - 35 (2006)
8. Data Encryption Standard, Federal Information Processing Standard (FIPS) Publication 46, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. (1977)
9. Advanced Encryption Standard (AES), Federal Information Processing Standard (FIPS) Publication 197, National Bureau of Standards, U.S. Department of Commerce, Washington D.C. (2001)
10. Matt J. B. Robshaw, Stream Ciphers Technical Report TR-701, version 2.0, RSA Laboratories, (1995)
11. Specifications for a Secure Hash Standard (SHS), Federal Information Processing Standards Publication DRAFT (1992)
12. Public Key Cryptography For The Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography, Draft ANSI X9F1, (1999)
13. NIST, Special Publication 800-22, A statistical test suite for random and pseudorandom number generators for cryptographic applications, (2001)
14. Aozora Bunko, http://www.aozora.gr.jp/
15. M. Matsumoto and T. Nishimura, Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator, ACM Transaction on Modeling and Computer Simulations, (1998)