論文 / 著書情報
Article / Book Information

Title	Consideration of Experimental Evaluationabout Encrypted Replica Update Process
Author	Kazuki Takayama, Dai Kobayashi, Haruo Yokota
Journal/Book name	Proceedings of the Second IEEE International Conference on Digital Information Management, , , pp. 545-550
Issue date	2007, 10
DOI	http://dx.doi.org/10.1109/ICDIM.2007.4444280
URL	http://www.ieee.org/index.html
Copyright	(c)2007 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other users, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works for resale or redistribution to servers or lists, or reuse of any copyrighted components of this work in other works.
Note	このファイルは著者（最終）版です。 This file is author (final) version.

Consideration of Experimental Evaluation about Encrypted Replica Update Process

Kazuki Takayama
Tokyo Institute of Technology
takayama@de.cs.titech.ac.jp

Dai Kobayashi
Tokyo Institute of Technology
Japan Society for the Promotion of Science
daik@de.cs.titech.ac.jp

Haruo Yokota
Tokyo Institute of Technology
yokota@cs.titech.ac.jp

Abstract

The secure storage systems adopting the encrypt-on-disk scheme, in which files are stored in cipher for efficient data transmission, need to re-encrypt files with new cryptographic keys when a revocation occurs. There are two re-encryption methods, namely active revocation in which the re-encryption is immediately performed and lazy revocation in which the re-encryption is delayed until the file is updated. There is the trade-off between performance and security because active revocation has the expense of immediate re-encryption, while lazy revocation is vulnerable during its re-encryption delay. We consider the environment in which re-encrypted file is pre-computed by using backup data in a parallel storage system effective for this issue. However, the performance of update is decreased on account of the difference of keys used in primary and backup. In this paper, we evaluate a method in which the differential data re-encrypted for backup are not written to the file but be kept on the memory in different key environment, and evaluate the different key environment in parallel storage by experiment.

1 Introduction

Storage systems have become networked. Because information security has recently gained increased recognition, data transmission in networked storage must be protected. Therefore, most secure storage systems include a data protection function[4, 5].

We have previously proposed an Autonomous Disk system[6], which is a dependable parallel storage system that autonomously executes such functions as fault tolerance, load balancing, and data distribution. The Autonomous Disk system comprises a cluster of highly functional disk nodes that are networked, but we have not previously considered data protection in the network.

We focus on confidentiality, one of the data protection requirements. Confidentiality refers to the property of a computer system whereby its information is disclosed only to authorized parties. It is commonly ensured by using cryptography. There are two fundamentally different approaches to using cryptography, namely the encrypt-on-wire scheme and the encrypt-on-disk scheme. The encrypt-on-disk scheme, by storing encrypted data, is more secure and has more efficient data transmission than the encrypt-on-wire scheme, which stores data in clear and encrypts them only when transmitted[4]. However, in the encrypt-on-disk scheme, data must be re-encrypted whenever users' access rights within the shared data environment are revoked. There are two re-encryption methods for revocation, active revocation and lazy revocation, and there is a trade-off between performance and security, in that active revocation may cause decreased performance because it must re-encrypt data immediately after a revocation, and lazy revocation may become vulnerable because it delays re-encryption until the next time the data are updated[4, 5].

We consider that we realize an efficient re-encryption for revocation where the encrypt-on-disk scheme is applied to a parallel storage system such as the Autonomous Disk system in which all accessed files are backed up on other disks. To solve the issue, we propose the environment where the backup data are encrypted with the keys differed from those used to the primary data. We make a prediction that the revocation processes are efficiently performed in the environment by pre-computing the re-encrypted data for revocation in backup. However, we also make a prediction that this environment costs more for update processes than the

environment where the same keys are used to the primary and backup data because the differential data must be re-encrypted in this processes on account of the key difference of the primary and backup data.

In this paper, we propose an efficient update method in the different key environment. When a file update occurs, the process of writing the differential data to the file is delayed and the differential data are kept on the memory. With this method, update can be performed at lower cost.

The remainder of this paper is organized as follows. Section 2 describes the revocation in the encrypt-on-disk scheme. Section 3 describes our proposal. Section 4 explains the assumptions and setup of the system, and Section 5 describes and evaluates our experiments. Finally, we present our conclusions in Section 6.

2 Revocation in the encrypt-on-disk scheme and issues

In this section, we describe the revocation processes in the encrypt-on-disk scheme where files are stored in cipher. At first, we explain the common cryptographic schemes for confidentiality in data transfer and the existing revocation methods in encrypt-on-disk scheme, active revocation and lazy revocation, and their problems. After that, we describe the strategy for resolving the problem, and the requirement of consideration for the strategy.

2.1 Encrypt-on-wire and encrypt-on-disk

In a networked storage system, there are two cryptographic schemes utilized for the protection of transmitted data from interception, namely the encrypt-on-wire scheme and the encrypt-on-disk scheme. The encrypt-on-wire scheme is a system whereby files are stored in clear and encrypted, using a key of secret-key cryptography created once per session, when transmitted. By contrast, the encrypt-on-disk scheme is a system whereby files are stored in cipher and transmitted without any encryption process.

From a view point of data transmission performance, the encrypt-on-disk scheme is more efficient than encrypt-on-wire scheme because the storage side of the encrypt-on-disk scheme may not have to do encryption and decryption work that the encrypt-on-wire scheme must do. In addition, the encrypt-on-wire schemescheme is at a disadvantage regarding key creation costs[4].

2.2 Revocation in the encrypt-on-disk scheme

In an encrypt-on-disk system where users share files, files must be re-encrypted with a new key if a revocation occurs. This is because revoked users may keep the current key, leading to information leakage if they intercept transmitted files, despite their access being denied by access control methods.

For this reason, the encrypt-on-disk scheme costs more for revocation than encrypt-on-wire scheme. However, Riedel et al. [4] provides survey results that showed that the encrypt-on-disk scheme offered both improved performance and increased security compared to the encrypt-on-wire scheme when the overall process is considered.

2.3 Existing methods for re-encryption

Existing re-encryption methods for the encrypt-on-disk scheme are divided into active revocation and lazy revocation according to the timing of re-encryption.

1. **active revocation** - it is a method that re-encrypts a file with a new key immediately after a revocation. This method is more secure than lazy revocation, because revoked users are immediately unable to decrypt the file. However, when a revocation occurs, a weak point is that, because files must be re-encrypt immediately, even authorized users cannot access the files until the re-encryption processes are complete. This weak point is more pronounced when multiple revocations occur at the same time for different files.

2. **lazy revocation** - it is a method that delays re-encryption until the file is next updated. It was first proposed in Cepheus[1], and was implemented in some secure storage systems, including Plutus[2]. Because update processes involve encryption, they can be combined with the re-encryption required for revocations. In addition, the re-encryption work for several revocations may be performed together if the file is not frequently updated. Therefore, in this case, lazy revocation is much more efficient than active revocation.

   However, lazy revocation is vulnerable. Its concept is that there should be no problem if revoked users access files before the update, because they may already have this information. Therefore, stored files before update are encrypted with the old key, which may be kept by revoked users. But there is the possibility that revoked users did not access the files before their access rights were revoked.

2.4 Issue and our strategy

Considering these security aspects, we should apply active revocation even though lazy revocation is a great deal better with respect to performance, but there is the performance problem in active revocation. The paper about

SNAD[3], one of the secure storage system adopting the encrypt-on-disk scheme, describes the revocation method as an issue for future work because of the trade-off problem.

To solve this issue, we consider the strategy that files which will be used after revocations are pre-computed by using backup data. Users can access immediately after the revocation by preparing the re-encrypted files. However, if the file updates occur in this environment, the performance decrement is predicted because the differential data for update should be re-encrypted on account of the backup data encrypted with the different keys from those used to primary, because the keys used to backup must be known by any users. Therefore, we propose the efficient process in this different key environment in this paper.

3 Approach

We propose the efficient processes for file update in the environment in which the backup data are encrypted with the keys used to primary data. In proposal, after re-encrypting the differential data with the key used to the backup data, which were encrypted with those used to the primary data, the differential data are not written to file but kept on the memory and delayed writing to file. By adopting this method, the update processes can be performed with lower cost than not adopting it, and the method do not spoil the merit on revocation as much as possible, that the files used after revocation are pre-computed by encrypting the backup data with the keys differed from those used to the primary data.

In this paper, we consider these environments:

1. **same key environment** - the backup data are encrypted with the same keys as those used to primary data. When a file is updated, the differential data are simply written to the primary data and the backup data because the same key is used. It is the environment of existing methods.

2. **different key environment** - the backup data are encrypted with the different keys from those used to primary data. When a file is updated, the differential data encrypted with the key used to primary data are re-encrypted with those used to backup data, and written to the backup data. It is the normal environment of our strategy.

3. **delayed writing environment** - its data structure is the same as the different key environment. When a file is updated, the differential data are re-encrypted with the backup key, but not written to backup data and kept in a table on the memory. It is the environment of the proposal in this paper.

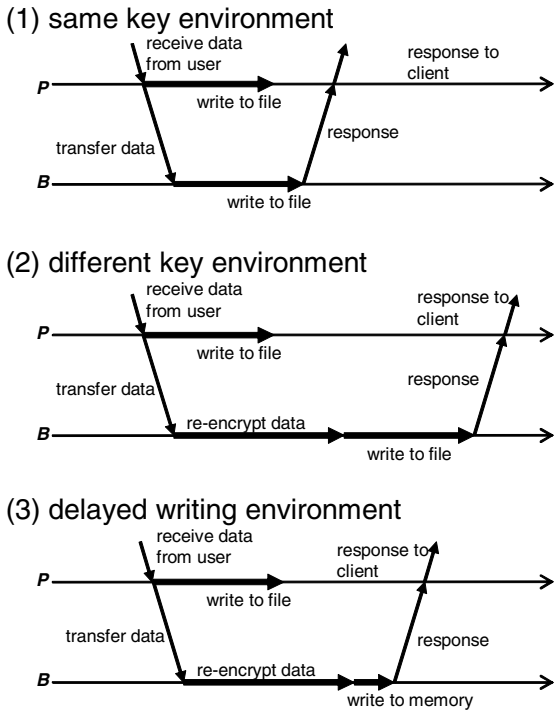

Figure 1. Workflow of update process for primary(P) and backup(B) data

The workflows of the update processes in each environment are shown in Figure 1.

4 System assumptions

4.1 Hardware construction

We propose to apply the encrypt-on-disk scheme to a parallel storage system or a cluster of file servers such as Autonomous Disks[6], which is a system which have previously proposed for managing data autonomously by utilizing processing function on storage devices. The Autonomous Disks implements the functions of fault tolerance, load balancing, and data distribution autonomously to reduce the management cost of storage administration. From this background, we assume that server side supports the processes of re-encryption and all storage servers are trusted to perform these processes.

In this paper, we assume the primary–backup structure implemented in these systems, whereby each disk node stores both primary data for user access and backup data, which replicates primary data stored in other disk nodes to implement the fault tolerance function. We leverage the backup data in this structure for our proposal.

We assume that each storage node is connected to a local network, and client nodes, which are also connected to the network, access these storage nodes via the network.

4.2 Use of cryptography

Cryptography is divided into the secret key cryptography and public key cryptography. In the secret key cryptography, the same key is used to encrypt and decrypt data, so a sender and a receiver must share the key in secure manner before data transmission. On the other hand, in public key cryptography, the keys for encryption and decryption are different, and together form a unique pair. In this system, there is no problem about key distribution because a user can expose one key, namely the public key, but there is the problem that the process of encryption is typically several hundred to a thousand times slower than for secret key cryptography. Generally, therefore, files are encrypted with a secret key and the secret key is encrypted with a public key for transmission because the keys are small and of fixed length. We will adopt this general technique, whereby files are encrypted with a secret key and the key is encrypted with a public key owned by users or storage nodes.

In this paper, we assume that the file is divided into one or more fixed blocks, and each block is separately encrypted to decrease the cost of update by defining the block as the unit of the differential data.

5 Experiments

We perform the experiment to test the performance of the proposal method. In this section, we describe the experimental programs and setup in 5.1, the experimental method in 5.2, and the results after those.

5.1 Setup for experiment

We created a server-client program to test the proposed method, which follows the encrypt-on-disk scheme and runs on a PC cluster. The server program builds three environments described in Section 3. By this server program, the backup data are made and stored in the storage node neighbored by that storing the primary data. The secret keys, with which files are encrypted, are encrypted with the public keys owned by the authorized users and stored with the files. By this concept, only authorized users can get files and keys. The client program can implement GET and UPDATE commands. In GET process, at first, a user sends the file name to server and receives the key encrypted with the user's public key and the encrypted file. Next, the user decrypts the file with the key and writes it to the client's disk. In UPDATE process, at first, a user sends the file name and the differential data encrypted with the primary key. The storage node storing the primary data of the update object file receives these data, and forwards them to the storage node storing its backup data, and writes the differential data to the primary data. The storage node of backup performs processes corresponding to the environment noted above. This UPDATE processes were detailed in Section 3 and Figure 1.

Table 1. Configuration of experimental system

CPU	AMD Athlon XP-M1800+ (1.53GHz)
Memory	PC2100 DDR SDRAM 1GB
HDD	TOSHIBA MK3019GAX(30GB/5400rpm/2.5inch)
Network	TCP/IP + 1000BASE-T
OS	Linux 2.4.20
Java VM	Sun J2SE SDK 1.5.0_03 Server VM

Table 2. Fixed parameters

Public-key algorithm	RSA 1024bit
Secret-key algorithm	AES 128bit
Encryption mode	ECB
Padding mode	PKCS5
Total primary data size	500MB/disk
Unit of differential data	250MB
Zipf parameter θ	0.9

The experiments were implemented on a PC cluster configuration, as described in Table 1, and used the parameter values given in Table 2. For the experiments, we fixed the total data size for a storage node, the size of each file, and the number of files stored in a storage node. The accessed files were selected according to a Zipf function to introduce bias. We selected the AES algorithm as the secret key algorithm and ECB mode as the encryption mode. AES is considered to be robust against, for example, known plaintext attack. ECB is the mode by which data are encrypted independently per block, and we used it to enable updates to be implemented by fixed-length blocks, as described in Section 4.2.

5.2 Methods of experiments

The experimental encrypt-on-disk system comprised three client nodes and three storage nodes. Previously, 500 1MB files are stored on each storage node. A client sends a GET of UPDATE request, which is determined by a variable parameter, to a storage node once every 400 ms, and the response times are measured. This interval was decided as the average response time of GET request in preparatory experiments. Under these conditions, we alter the GET-UPDATE ratio and observed the response times in two requests independently.

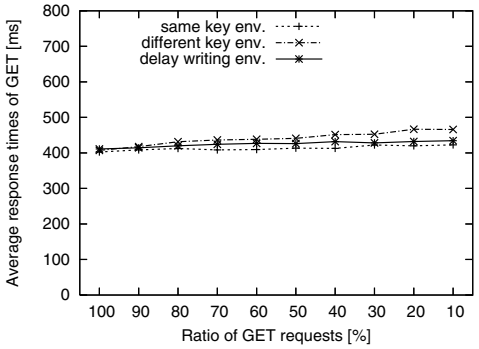

Figure 2. Average response times of GET

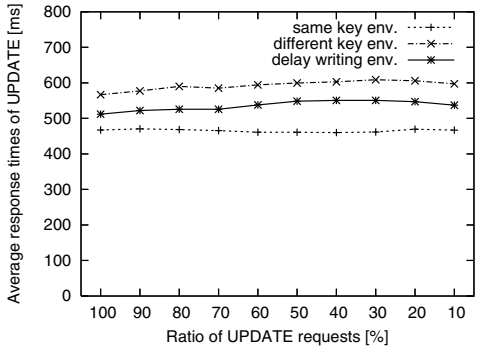

Figure 3. Average response times of UPDATE

We experimented for three minutes in each request ratio and performed the series of experiments five times, and calculated the average in all experiments and all nodes.

5.3 Results and discussion about average response times

Figure 2 and 3 show the average response times of each request under the each request ratio.

These two figures show that the average response times of GET and UPDATE requests in delayed writing environment are shorter than those in the different key environment. On the other hand, the response times in delayed writing environment are longer than those in same key environment, but we consider that they are sufficiently comparable, particularly about the GET requests.

GET processes are same in all environments, but those in a storage node are affected by the processes in backup of the node performed by the UPDATE processes in other storage node, so the average response times are different. The results show that re-encryption of the differential data has larger effect than writing those to file, and the different key environment in which these two processes are performed has the worst performance in GET process.

The performance of UPDATE processes depends essentially on the amount of times taken in backup processes. The result that the same key environment has more improved performance than the delayed writing environment shows that re-encryption of differential data requires time more than writing those to file.

5.4 Results and discussion about average of maximum response time

Figure 4 and 5 show the average of maximum response times of each request.

These figures show that the maximum response times in the delayed writing environment are the shortest in those

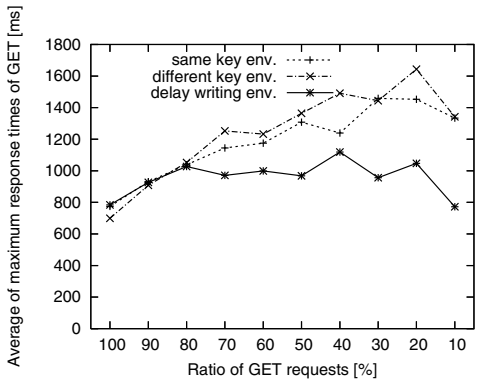

Figure 4. Average of maximum response times of GET

of all environments. We consider that the factor of this result is that there is no process of writing differential data to file in the delayed writing environment. In the same key environment and the different key environment, the backup processes of UPDATE include the writing process, and if this occurs at the same time as the process of reading file in GET processes or writing differential data to file in UPDATE processes occurring in primary, they have possibilities to be made to wait. On the other hand, in the delayed waiting environment, the process of re-encrypting the differential data in backup and the process of reading file of writing differential data to file in primary do not interfere with each other even if they occur at the same time.

5.5 Discussion

In the results about average response times, the delayed writing environment has more improved performance than the different key environment, but is slightly inferior to the same key environment. On the other hand, in the results

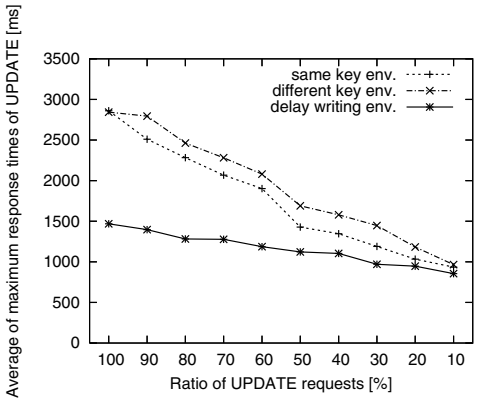

Figure 5. Average of maximum response times of UPDATE

about maximum response times, the delayed writing environment has the most improved performance in all environments. Considering the performance in the environment in which revocations occur in addition to the performances described above, we estimate that the performance in the delayed writing environment do not get worth comparatively even if the system is in the worst state, while the performances in the other two environments may take significant performance degradations.

In the delayed writing environment, the differential data kept on the memory and the file are inconsistent. Therefore we plan to consider the timing of writing them to the file because it has a possibility to make the system performance decreace.

6 Conclusions and future work

In this paper, we present the efficient file update method in the environment of the primary–backup structure in which the backup data are encrypted with the keys differed from those used to primary data. The delayed environment adopting this method have more improved performance about the average response times of GET and UPDATE requests than the different key environment, and have the most improved performance about the maximum response times in the three environments including the same key environment.

We assume that the revocation processes are efficiently performed in the environment in which the backup data are re-encrypted with the keys differed from those used to the primary data because the data used after a revocation occurs are pre-computed. Considering the assumption of the performance when a revocation occurs and the result in this paper, we consider that the delayed writing environment does not have the lowest performance anytime even if the system is in the worst status, while the others may sometimes have the terrible performance.

In future, we should perform the experiments in this paper including the situation in which revocations occur, and evaluate the comprehensive performances of three environments. In addition, we should think the demerit caused by the situation that the primary data and the backup data are encrypted with the keys differing each other in the delayed writing environment.

Meanwhile, we must consider the strategy of writing the differential data kept on the memory which is delayed in the delayed writing environment. We assume that this process does not make the performance decrease if that is performed when the system is idle, and so we should implement and evaluate it.

After addressing these issues, we plan to implement the proposed method on Autonomous Disks, and evaluate the effectiveness of the proposal in a practical environment.

Acknowledgment

This research is sponsored by CREST of JST (Japan Science and Technology Agency), the SRC, the NHK Science and Technical Research Laboratories, MEXT via Grant-in-Aid for Scientific Research #19024028, the Tokyo Institute of Technology 21COE Program “Framework for Systematization and Application of Large-Scale Knowledge Resources”, and the Japan Society for the Promotion of Science via Gran-in-aid for JSPS Fellows.

References

[1] K. Fu. Group sharing and random access in cryptographic storage file system. Master’s thesis, MIT, 1999.

[2] M. Kallahalla, E. Riedel, R. Swaminathan, Q. Wang, and K. Fu. Plutus: Scalable Secure File Sharing on Untrusted Storage. In *FAST ’03: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, pages 29–42. USENIX Association, 2003.

[3] E. Miller, D. Long, W. Freeman, and B. Reed. Strong Security for Network-Attached Strorage. In *FAST ’02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, pages 1–13, Berkeley, CA, USA, 2002. USENIX Association.

[4] E. Riedel, M. Kallahalla, and R. Swaminathan. A framework for evalating storage system security. In *FAST ’02: Proceedings of the 1st USENIX Conference on File and Storage Technologies*, pages 15–30. USENIX Association, 2002.

[5] P. Stanton. Securing Data in Storage: A Review of Current Reserch. *ArXiv Computer Science e-prints*, 2004.

[6] H. Yokota. Autonomous Disks for Advanced Database Applications. In *Proc. of International Symposium on Database Applications in Non-Traditional Environments (DANTE’99)*, pages 435–442, Nov. 1999.