Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2014, Article ID 293214, 8 pages
http://dx.doi.org/10.1155/2014/293214

Research Article

An Enhanced Erasure Code-Based Security Mechanism for Cloud Storage

Wenfeng Wang,[1,2] Peiwu Li,[1,2,3] Longzhe Han,[1,2] Shuqiang Huang,[4] Kefu Xu,[5] Changgui Yu,[2] and Jin'e Lei[1,2]

[1] *Institute of Computer Network & Information Security, Nanchang Institute of Technology, Nanchang 330099, China*
[2] *School of Information Engineering, Nanchang Institute of Technology, Nanchang 330099, China*
[3] *Department of Scientific Research, Nanchang Institute of Technology, Nanchang 330099, China*
[4] *Network and Education Technology Center, Jinan University, Guangzhou 510632, China*
[5] *Institute of Information Engineering, Chinese Academy of Sciences, National Engineering Laboratory for Information Security Technologies, Beijing 100093, China*

Correspondence should be addressed to Wenfeng Wang; wangwf@nit.edu.cn

Received 1 July 2014; Accepted 11 September 2014; Published 16 November 2014

Academic Editor: Yuxin Mao

Cloud computing offers a wide range of luxuries, such as high performance, rapid elasticity, on-demand self-service, and low cost. However, data security continues to be a significant impediment in the promotion and popularization of cloud computing. To address the problem of data leakage caused by unreliable service providers and external cyber attacks, an enhanced erasure code-based security mechanism is proposed and elaborated in terms of four aspects: data encoding, data transmission, data placement, and data reconstruction, which ensure data security throughout the whole traversing into cloud storage. Based on the mechanism, we implement a secure cloud storage system (SCSS). The key design issues, including data division, construction of generator matrix, data encoding, fragment naming, and data decoding, are also described in detail. Finally, we conduct an analysis of data availability and security and performance evaluation. Experimental results and analysis demonstrate that SCSS achieves high availability, strong security, and excellent performance.

1. Introduction

High performance, rapid elasticity, on-demand self-service, and low cost, all these luxuries contribute to the popularity of cloud computing [1, 2]. As a new delivery and consumption model of IT services, cloud computing provides incomparably super computing power and huge storage capacity via the Internet. Now, many well-known companies, such as Amazon, Google, Microsoft, and Alibaba, have established their cloud platforms, respectively, to provide various online services. Meanwhile, being the infrastructure of cloud computing, cloud storage [3, 4] has also received extensive attentions from both academic and industrial fields and become a worldwide research hotspot.

Simply, cloud storage is considered as an Internet-based super storage model. Users can access cloud storage platforms by PCs, laptops, mobile phones, and other portable devices anytime and anywhere. Compared with traditional storage model, cloud users only need to pay a small fee but can enjoy unlimited storage spaces and ubiquitous storage services. However, on one side, cloud storage servers may not be completely trustworthy and data may be revealed by service providers because of economic interests and other factors. On the other side, no matter trade secrets or individual privacies, these important data are not only available to cloud users but also to service providers, while users become incapable of controlling their own data due to the separation of data management and ownership in cloud. As a result, some data security threats appear [5]. That is why enterprises are reluctant to deploy their businesses in the cloud even though cloud computing provides a wide range of luxuries. In fact, data security has become one of the major issues which acts

as an obstacle in the promotion and popularization of cloud computing [6].

Currently, erasure code is widely used in existing cloud storage systems, in which data usually have to go through several procedures, such as data encoding, data transmission, data placement, and data reconstruction. That means the security mechanism should be provided at each process accordingly. Previous researches have paid much attention to the issues of data security in distributed storage systems. However, few of them address the security issues in all of the above-mentioned stages. Therefore, they cannot well satisfy the secure requirements in cloud storage. In this paper, an enhanced erasure code-based security mechanism for cloud storage is presented, which provides data protection in each process of data access. The mechanism can effectively solve the problem of data leakage resulted from incredible cloud service providers and external cyberattacks. On the basis of this mechanism, we implement a secure cloud storage system. It not only offers high data availability, but also ensures strong data security.

The rest of the paper is organized as follows. Section 2 introduces some related work. Section 3 provides an overview of the cloud storage model. Section 4 proposes the erasure code-based security mechanism for cloud storage. The key design issues are elaborated in Section 5, where we describe the data division, construction of generator matrix, data encoding, fragments naming, and data decoding. Performance evaluation and analysis is presented in Section 6. Finally, Section 7 summarizes our conclusions.

2. Related Work

High availability and strong security are two important properties that most cloud storage services offer today. Replication and erasure code are the two common methods used to achieve high availability [6–8]. Ma et al. [9] proposed a novel scheme for cloud file system which brought together replication and erasure code. The authors in [10] introduced a cloud storage system named MassCloud. The availability of MassCloud was guaranteed by a strategy combining erasure code, replication, and RAID. Both of them adopted replication and erasure code to varying degrees and offered high availability. However, replication scheme has several deficiencies in data security. First, any intrusion into a replica host node can acquire the data; as to erasure code, it has to concurrently invade multiple nodes. Second, to achieve equivalent data availability, the former needs to consume much more storage resources. Even worse, the more the replicas are created, the easier the data are to be exposed. This further worsens the data security. Third, data may be leaked by cloud service providers and external cyberattacks; even a user realizes the potential risks, he/she cannot destroy all the replicas stored in cloud to minimize the loss. That is to say, cloud user is put in a passive position in this case. As a matter of fact, cloud storage systems are transitioning from replication to erasure code [11, 12]. That is why our security mechanism is designed on the basis of erasure code.

To tackle the problem of data leakage from internal threats in cloud, Xu et al. [13] presented a novel data privacy protection mechanism which partitioned the original data into two blocks: a small one which was deployed locally and a large one which was deployed remotely. This mechanism can resist data breach effectively caused by untrustworthy service providers, but there still exist the following weak points. (1) As all small blocks are stored locally, there is an enhancement in the complexity of data management as well as a reduction in the data availability. (2) Data recovery must be executed in the host nodes of small blocks, which is contrary to the original intention of accessing cloud data anytime and anywhere. This may be unacceptable to cloud users.

Data security issues have also been well studied in the literatures [14–17]. Prasad et al. [14] proposed a model by utilizing data classification and 3-dimensional authentication approaches. However, since data was stored in plain text in this model, once a user lost his/her credential, the data could be obtained by any unauthorized user. Hwang and Li [15] adopted data coloring method to secure data access at a fine-grained file level. Yang and Jia [16] introduced a privacy-preserving third-party auditing protocol to check the data integrity in cloud. Sood [17] proposed a model by incorporating several techniques, such as SSL (secure socket layer) and MAC (message authentication code), to ensure data security in cloud. It is worthy to point out that data classification, data coloring, third-party audit, and other policies can be combined with the mechanism proposed in this paper to strengthen data security in cloud.

3. The Cloud Storage Model

3.1. Three Functional Components. As shown in Figure 1, the cloud storage model revolves around the following functional components: cloud metadata server (CMS), cloud storage server (CSS), and cloud client (CC).

In SCSS, all metadata information is maintained at CMS. More often, the CMS is utilized to make a strategic decision on many tasks, such as management of CCs and CSSs, data placement, fault tolerance and recovery, and load balancing. By the way, although the CMS is logically unique, it can be physically organized in master/slave, cluster, P2P, or other architectures for scalability and availability. As is implied by the name, CSS is a software server that mainly performs data storage. Generally, data file is split into stripes of several blocks and delivered to different cloud storage clusters. CC is an entity which has critical data files to be backed up to and recovered from cloud. It relies on cloud for data availability and security.

3.2. Process of Accessing Cloud Data. Figure 1 depicts the process of data access in cloud. When a client tries to access cloud data, it firstly sends a request to the CMS (Step 1). Then, the CMS validates the user's credential (Step 2). If the user is authenticated, then the CMS will send some messages to the client and relative CSSs, respectively (Step 3). After that, the client interacts with the appropriate CSSs directly without intervention of the CMS (Step 4). Finally, the client reports

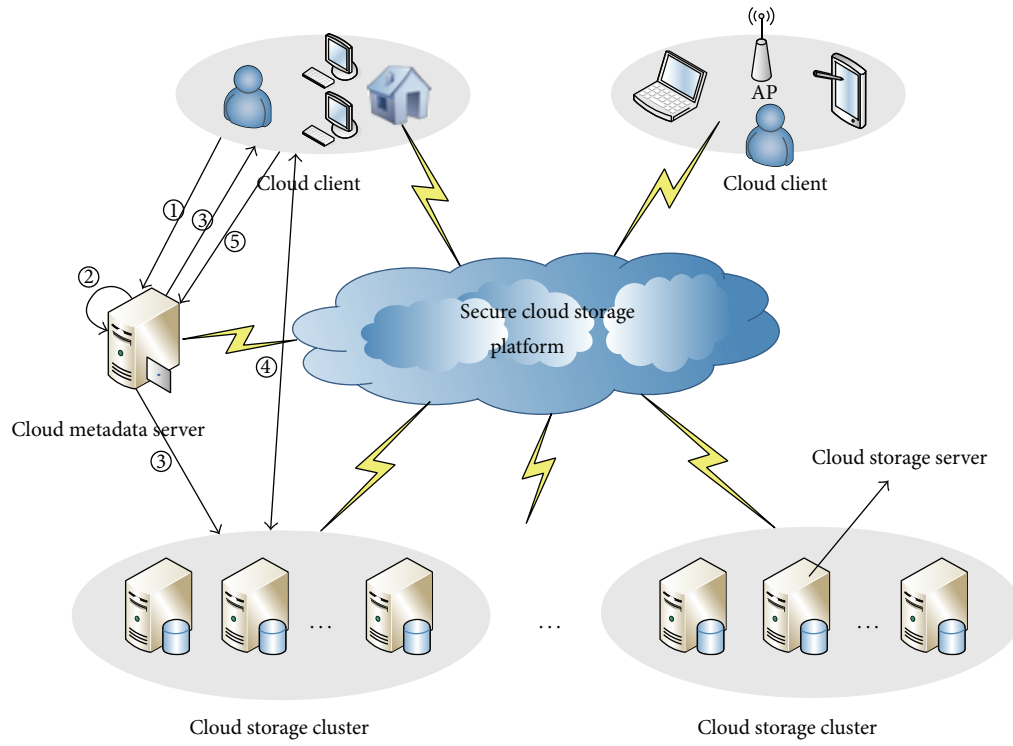

Figure 1: Three functional components of the cloud storage model.

its fulfillment to the CMS (Step 5). As data is spread across multiple CSSs, it can be accessed simultaneously, which is manifested in improved system performance.

4. Erasure Code-Based Security Mechanism for Cloud Storage

4.1. Reed-Solomon Erasure Code. Given an original data file, Reed-Solomon erasure code [6–8] firstly divides it into k fragments of the same size and then encodes them into n fragments. Any k fragments taken out of the n encoded ones can be used to reconstruct the original data file. Meanwhile, it is impossible to obtain any information about any fragment of the original data from less than k fragments. Therefore, Reed-Solomon code supports k-resistance and ensures high security. This code is customarily referred to as (n, k) Reed-Solomon code.

Mathematically, (n, k) Reed-Solomon code can be expressed as

$$S = G \cdot F. \tag{1}$$

In the above equation, $F = \{F_i \mid 1 \le i \le k\}$ is the original data file, $S = \{S_i \mid 1 \le i \le n\}$ is the encoded data file, where F_i and S_i are the corresponding row vectors of the ith fragment of F and S, respectively, and G is a generator matrix of (n, k) linear Reed-Solomon code.

Theorem 1. *Let G be a generator matrix of a (n, k) linear Reed-Solomon code. If any k square submatrix of G is invertible, then any k encoded data fragments are sufficient to reconstruct the original data file.*

Proof. Let S^* be the row vectors consisting of any k elements of the encoded data file S (i.e., S^* is composed of any k encoded data fragments) and G^* the corresponding k square submatrix of G. Obviously, we can infer from (1):

$$S^* = G^* \cdot F. \tag{2}$$

From the assumption, since the submatrix G^* is linearly independent, then

$$F = \left(G^*\right)^{-1} \cdot S^*. \tag{3}$$

According to the above equation, the original data file F can be reconstructed. □

4.2. Enhanced Erasure Code-Based Security Mechanism. As shown in Figure 2, the major design philosophies of the enhanced erasure code-based security mechanism can be described as below.

Step 1. Divide the original data file F into k fragments $(F_1, F_2, \ldots, F_k)$.

Step 2. Encode these fragments into n fragments $(S_1, S_2, \ldots, S_k, S_{k+1}, \ldots, S_n)$.

Step 3. Encrypt the above n fragments into encrypted ones $(E_1, E_2, \ldots, E_k, E_{k+1}, \ldots, E_n)$ and distribute them to n different storage nodes.

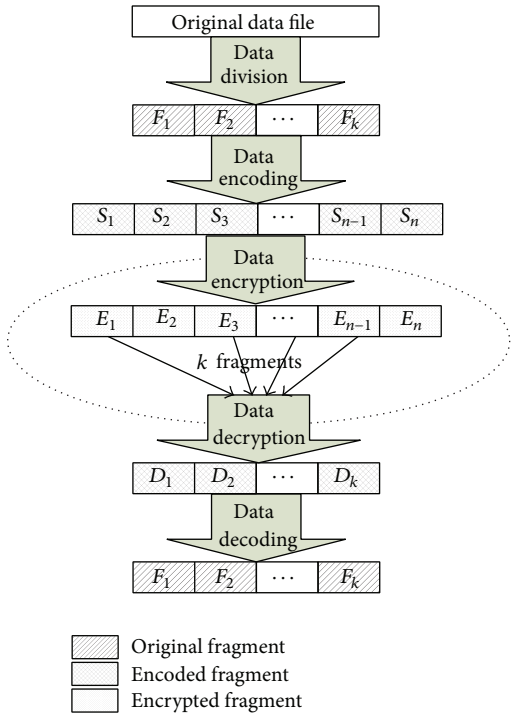

Figure 2: Enhanced erasure code-based security mechanism.

Step 4. Take any k fragments out of the n encrypted ones and decrypt them into k fragments $(D_1, D_2, \ldots, D_k)$ in plain text.

Step 5. Reconstruct the original data file F by using (3).

From the above description, cloud data has to go through different steps, so the security mechanism should be provided at each stage. We will discuss the issues in detail in the subsequent sections.

4.2.1. Data Encoding Security

(1) *Transparency of the fragment number.* When an original file is under dividing, the value of k will be produced automatically and thoroughly transparent to users under the condition that it meets users' requirements. Consequently, it will be difficult for attackers to know how many fragments are needed to reconstruct the original data, which enhances the difficulty of data reconstruction.

(2) *Randomness and encryption of the generator matrix.* We employ random function to yield the generator matrix and encrypt it at the same time. Hence, even if k relative fragments have been collected, the original data still cannot be reconstructed for an absence of the generator matrix.

(3) *Irrelevance of fragment naming.* We use hash function to deal with the naming of encrypted fragments. Thus, it is impossible for attackers to determine the corresponding sequence of each fragment by its name from a great number of fragments; therefore, they are bound to fail to acquire a submatrix strictly in sequence. Meanwhile, due to the irrelevance of fragment naming, it is almost impossible for attackers to spot k relevant fragments from massive fragments in cloud storage.

4.2.2. Data Transmission Security. When data fragments are being transferred from a client to multiple CSSs, they are under the risk of being tampered or intercepted by malicious users. As mentioned above, we have borrowed the ideas of SSL and MAC [17] to ensure data integrity and confidentiality in our cloud storage system.

4.2.3. Data Placement Security. On one hand, we try to select n different storage clusters which are geographically dispersed and systematically heterogeneous, with each cluster being responsible for storing one fragment. On the other hand, as to each storage cluster, we employ hash, round-robin, or other strategies to select proper host nodes. As a result, attackers must at first succeed in intruding k heterogeneous clusters and then search for the host nodes of fragments from a wide range of clouds. The dispersal and heterogeneity undoubtedly enhance the difficulty of fragments to be stolen.

4.2.4. Data Reconstruction Security

(1) *Session key.* When a user requests to recover some data files, the system yields a session key randomly, which will be used for the communication between the user and the host nodes of k fragments. For this key contains a specific validity, it can efficiently prevent attackers' continuous attempt of intruding the system for a long time even if it has been captured.

(2) *Data decoding key.* As shown in Figure 2, all the fragments have been encrypted before being delivered to host nodes. Thus, even though an attacker has collected k fragments, he/she cannot fulfill data decryption and the follow-up data reconstruction due to a lack of the key.

(3) *Constraints of access IP and time.* For those files which require extremely high-level security, such as business secrets and individual privacies, we can limit the data access by setting up specific IP addresses or time periods. In other words, higher security can be achieved at the expense of losing space and time flexibility to some extent.

5. Key Design Issues

On the basis of the aforementioned mechanism, we implement a secure cloud storage system (SCSS). SCSS is implemented in C++ (approximately 20,000 lines of code) and currently deployed in the campus network for hosting secure data storage services. We will present the key design issues in this section.

5.1. Data Division. Suppose that the source file F needs to be divided into k fragments of the same size, and each fragment is split into multiple segments with the size of *StepSize* bytes. To make the file size of F be exactly the integer multiple of $k \times StepSize$, whether a filling is needed depends on the real situation. We denote the filled file size as *fileSize,* and then the size of each fragment is $fileSize/k$. In accordance with the principle of data splitting, each fragment will be split into m segments ($m = fileSize/k \times StepSize$). Let f_{ij} be the jth segment of the ith fragment; then each fragment can be denoted as an m-dimension row vector F_i $(f_{i1}, f_{i2}, \ldots, f_{im})$, and the source file F can be expressed as the following $k \times m$ matrix:

$$F = \left(f_{ij}\right)_{k\times m} = \begin{pmatrix} f_{11} & f_{12} & \cdots & f_{1m} \\ f_{21} & f_{22} & \cdots & f_{2m} \\ \vdots & \vdots & \cdots & \vdots \\ f_{k1} & f_{k2} & \cdots & f_{km} \end{pmatrix}. \tag{4}$$

5.2. Construction of Generator Matrix. As previously mentioned, with regard to a generator matrix of (n, k) linear Reed-Solomon erasure code $G = (g_{ij})_{n\times k} = \begin{pmatrix} g_{11} & g_{12} & \cdots & g_{1k} \\ g_{21} & g_{22} & \cdots & g_{2k} \\ \vdots & \vdots & \cdots & \vdots \\ g_{n1} & g_{n2} & \cdots & g_{nk} \end{pmatrix}$, any $k \times k$ submatrix G^* must be invertible. To satisfy the above requirement, previous research usually adopts *Vandermonde* or *Cauchy* matrix. However, there are some practical limitations; for instance, the value of k is too big to be suitable for small files. Here, we introduce a simple method to yield the generator matrix. We firstly employ the combination of random functions *srand*() and *rand*() to generate an $n \times k$ matrix automatically and then check the matrix to ensure its linear irrelevance, where *srand*() is used to initialize random number generator, ensuring that every generator matrix is different, and *rand*() is used to produce different elements of a generator matrix. An interesting phenomenon was found in our mass tests (the procedure of matrix generation runs 100,000 times, with the value of k ranging from 3 to 20 and n being two times of k), all the $k \times k$ submatrixes of the generator matrixes produced by employing the above method could ensure the linear irrelevance. The value range of the elements of the matrix will be explored in the next section.

5.3. Data Encoding. As can be seen from (1), the matrix of the encoded data file S can be gotten by multiplying the matrix of the source data file F and the generator matrix G as follows:

$$S = \left(s_{ij}\right)_{n\times m} = \begin{pmatrix} \sum_{i=1}^{i=k} g_{1i} \cdot f_{i1} & \sum_{i=1}^{i=k} g_{1i} \cdot f_{i2} & \cdots & \sum_{i=1}^{i=k} g_{1i} \cdot f_{im} \\ \sum_{i=1}^{i=k} g_{2i} \cdot f_{i1} & \sum_{i=1}^{i=k} g_{2i} \cdot f_{i2} & \cdots & \sum_{i=1}^{i=k} g_{2i} \cdot f_{im} \\ \vdots & \vdots & \cdots & \vdots \\ \sum_{i=1}^{i=k} g_{ni} \cdot f_{i1} & \sum_{i=1}^{i=k} g_{ni} \cdot f_{i2} & \cdots & \sum_{i=1}^{i=k} g_{ni} \cdot f_{im} \end{pmatrix}. \tag{5}$$

In the above matrix S, each element is a segment and each row vector is an encoded fragment.

Taking the efficiency and complexity of implementation into consideration, the value of *StepSize* takes 3; that is, each segment takes 3 bytes. We use a variable of unsigned long type to store a segment and set the high byte zero. Let *MaxStep* be the maximum value (2^{24}-1) of the variable. We also use another variable of long type to store an encoded segment and let *MaxExtend* be the variable's maximum value (2^{31}-1). Thus we can get $MaxExtend/MaxStep \approx 128$.

Now, we will investigate the value range of the elements of a generator matrix. Taking the element s_{11} of the matrix S as an example, the following inequation always holds only if $g_{1i} \cdot f_{i1} \leq MaxExtend/k$:

$$s_{11} = \sum_{i=1}^{i=k} g_{1i} \cdot f_{i1} \leq MaxExtend. \tag{6}$$

Since the maximum value of f_{i1} is *MaxStep*, if $g_{1i} \leq MaxExtend/k \times MaxStep$, then the above inequality is constantly valid. Hence, the maximum value of an element of a generator matrix should be less than $MaxExtend/k \times MaxStep \approx 128/k$. Meanwhile, the elements on the diagonal of the matrix cannot be zero, so the value range of the elements of a generator matrix is $\{x \mid x \in [-\text{Floor}(128/k), \text{Floor}(128/k)] \wedge x \in Z \wedge x \neq 0\}$; here Floor means rounding down.

5.4. Fragments Naming. To prevent unauthorized users from obtaining enough relative fragments and the corresponding submatrix G^* strictly in sequence, we use the follow-up method to produce fragments' names:

$$\begin{aligned} fn_i &= \text{Hash}\left(fn_{i-1} \| sk \| v\right), \quad 1 \leq i \leq k, \\ fn_0 &= \text{""}, \end{aligned} \tag{7}$$

where fn_i denotes the filename of the ith fragment, sk is the session key as described in Section 4.2.4, and v is a 128 bit vector, which can be an amalgam of some properties of the original data file, such as file size, suffix, and creation time.

5.5. Data Decoding. Equation (3) indicates that data decoding can be achieved once the corresponding submatrix of k fragments is invertible. Given k fragments and the corresponding submatrix, the key issue of data decoding is to figure out the inverse matrix $(G^*)^{-1}$. However, we find that it will result in a huge amount of overhead if we solve the linear equations (3) in such a way (for simplicity, we call this way as matrix inversion). Here the formula of Gauss elimination is adopted in our implementation. We will demonstrate the results in the next section.

6. Experiments and Analysis

6.1. Data Availability and Security Analysis. To clearly illustrate the data availability of replication and erasure code, suppose that the original data file size is 1 Mbytes, the

Table 1: Comparison of data security.

Security items	Chen and He [12]	Xu et al. [13]	Proposed mechanism
Data encoding	Low	Medium	High
Data transmission	—	—	High
Data placement	Medium	Low	High
Data reconstruction	Medium	High	High
Resistance of internal data leakage	Medium	High	High

Table 2: Comparison of data decoding time.

File size = 4 MB	Matrix inversion (ms)	Gauss elimination (ms)	File size = 32 MB	Matrix inversion (ms)	Gauss elimination (ms)
$k = 3, n = 20$	328	67	$k = 3, n = 20$	2644	536
$k = 4, n = 20$	488	68	$k = 4, n = 20$	3913	547
$k = 5, n = 20$	740	74	$k = 5, n = 20$	5823	596
$k = 6, n = 20$	1074	79	$k = 6, n = 20$	8624	658
$k = 7, n = 20$	1510	85	$k = 7, n = 20$	12107	698
$k = 8, n = 20$	2087	89	$k = 8, n = 20$	16663	713
$k = 9, n = 20$	2719	94	$k = 9, n = 20$	21841	800
$k = 10, n = 20$	3641	101	$k = 10, n = 20$	28939	821

available redundant storage capacity is also 1 Mbytes, and the average failure rate of a storage node is 10%. In this case, the data availability under replication scheme which has two replicas is 99%. As to erasure code, when $k = 4, n = 8$, the data availability is 99.96%; when $k = 5, n = 10$, it can offer four 9's of data availability.

From the above description, we know that any malicious user who wants to obtain the original data file must satisfy the following requirements: (1) searching k relevant data fragments from irrelevantly named massive ones (k is unknown); (2) intruding k standalone storage nodes and getting these fragments; (3) cracking k AES encrypted data fragments; (4) breaking the generator matrix and finding the strictly corresponding sub matrix. Therefore, it is impossible to accomplish all these tasks in a limited time. That means we can ensure the security of the data file.

Table 1 shows the comparison of the proposed mechanism with other strategies. It indicates that the enhanced erasure code-based mechanism has provided a strong data protection policy which covers all the stages of data access and attains a high-level data security. Although the method proposed by Xu et al. [13] can effectively resist data leakage due to internal threats from cloud service providers, it has a number of deficiencies as mentioned earlier.

6.2. Performance Evaluation. We firstly compare the data decoding time of matrix inversion with that of Gauss elimination. The value of k ranges from 3 to 10, $n = 20$, and the file size takes 4 MB and 32 MB, respectively. As shown in Table 2, the data decoding time of the former is much longer than that of the latter. With the increase of k, the data decoding time of matrix inversion grows dramatically, which rises nearly ten times. As to that of Gauss elimination, it sustains a steady increase; that is, the parameter k has a little impact on the decoding time. That means we can flexibly adjust the value of k to well satisfy users' requirements.

Now, we investigate the influence of different fragment numbers on the time of data backup and data recovery. In our tests, 120 storage servers are evenly partitioned to 10 clusters. All these storage servers are homogeneous and connected via a 1 Gbps Ethernet switch. Each storage node is equipped with Pentium dual-core CPU 2.93 GHz, 2 GB memory, and 300 GB 7200 RPM hard disk. We also adopt another two servers with the same configurations and use them as the metadata server and the client. Data backup time mainly consists of four parts: I/O, data encoding, fragments transmission, and the others. Similarly, data recovery time is also composed of four components: I/O, data decoding, fragments transmission, and the others. Obviously, either data backup time or data recovery time grows with the increase of the encoded file size, which is determined by the ratio of n/k. Consequently, the ratio is constantly set as two in our tests (i.e., $n = 2k$), the value of k ranges from 3 to 10, and the file size takes 4 MB. We use AES for data encryption.

Figure 3 illustrates the relationship between data backup time and fragment numbers. It can be seen that data backup time declines steadily in the beginning and goes down to the lowest point of 619 ms when k takes 6. Since then, it sustains an upward trend with the increase of k. As shown in Figure 4, data recovery time changes in a similar way as that of data backup time. However, the former is always much shorter than the latter. One of the main reasons is that only k fragments are enough for data recovery, but n fragments are requisite for data backup. It can be also observed that fragments transmission time plays an important role in both the data backup time and data recovery time. And the effects of the other three components are small and variable. As data

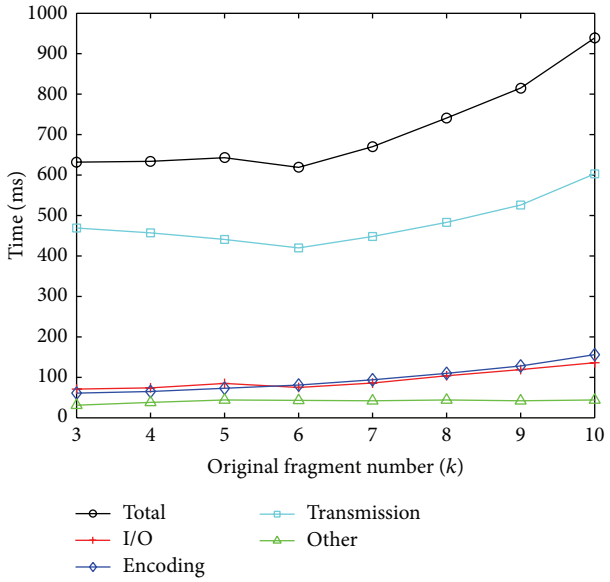

Figure 3: Data backup time under different fragment numbers.

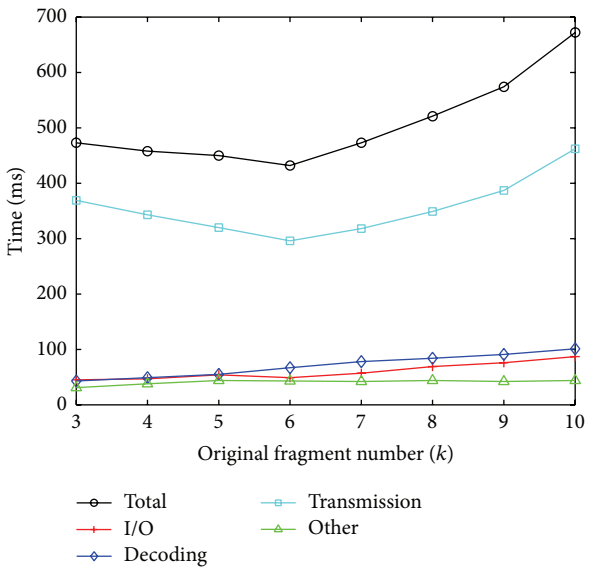

Figure 4: Data recovery time under different fragment numbers.

backup time and data recovery time are less than 1,000 ms, the speed of the data encoding as well as data decoding is far more than 4 MB/s. That means our proposed mechanism obtains an excellent performance.

Furthermore, we also evaluate the extra overhead introduced by the proposed mechanism compared to traditional erasure code. The value of k and n takes 4 and 8, respectively, and the file size increases exponentially from 1 MB to 512 MB. Similarly, AES is adopted to encrypt the data fragments.

Figure 5 presents the percentages of the extra overhead relative to the cost of traditional erasure code under different file sizes. The percentage firstly sustains an increasing trend and reaches a peak of 2.64% when the file size is 16 MB. Since then, it flats at 2.55% and finally keeps almost the same. It is

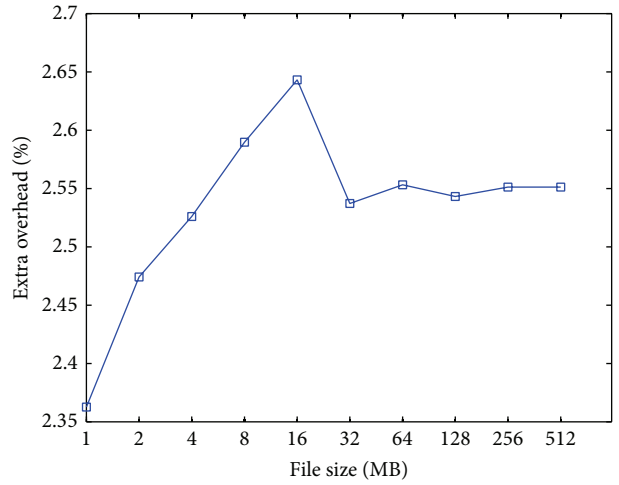

Figure 5: Percentages of extra overhead under different file sizes.

obvious that our proposed mechanism has little influence on the system overhead, with a tiny increase between 2.36% and 2.64%. Therefore, the extra overhead owing to the mechanism is negligible and acceptable in practical application.

7. Conclusions

In this paper, we mainly focus on the problem of data leakage owing to unreliable service providers and external cyberattacks in cloud storage. We firstly provide an overview of the secure cloud storage model. Then, we propose an enhanced security mechanism based on erasure code and elaborate it from four aspects: data encoding, data transmission, data placement, and data reconstruction. Finally, we present the key design issues and implement a prototype SCSS. Experimental results and analysis demonstrate that SCSS achieves high availability, strong security, as well as excellent performance. As data is stored in ciphertext in SCSS, how to effectively retrieve files from cloud by searching over encrypted data is a question that is worthy to be studied in the future.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (nos. 60573145 and 61373125), Jiangxi Natural Science Foundation (nos. 2010GQS0165 and 20132BAB201032), Jiangxi Science and Technology Supporting Program Foundation (nos. 20122BBE500050 and 20132BBE50047), Jiangxi Educational Committee Science and Technology Program Foundation (nos. KJLD13095 and GJJ14750), Educational Commission Fund of Guangdong Province (no. 2013KJCX0018), Guangdong Science

and Technology Plan Foundation (nos. 2012B010100027 and 2012B091100161), and Nanchang Science and Technology Supporting Program Foundation (no. 2012-CYH-DW-XX@GGAQ-002).

References

[1] S. Ma, "A review on cloud computing development," *Journal of Networks*, vol. 7, no. 2, pp. 305–310, 2012.

[2] S. Misra, P. V. Krishna, K. Kalaiselvan, V. Saritha, and M. S. Obaidat, "Learning automata-based QoS framework for cloud IaaS," *IEEE Transactions on Network and Service Management*, vol. 11, no. 1, pp. 15–24, 2014.

[3] Cloud Storage, 2014, http://en.wikipedia.org/wiki/Cloud_storage.

[4] J. Wu, L. Ping, X. Ge, W. Ya, and J. Fu, "Cloud storage as the infrastructure of cloud computing," in *Proceedings of the International Conference on Intelligent Computing and Cognitive Informatics (ICICCI '10)*, pp. 380–383, Kuala Lumpur, Malaysia, June 2010.

[5] L. M. Kaufman, "Data security in the world of cloud computing," *IEEE Security and Privacy*, vol. 7, no. 4, pp. 61–64, 2009.

[6] J. Idziorek and M. Tannian, "Security analysis of public cloud computing," *International Journal of Communication Networks and Distributed Systems*, vol. 9, no. 1-2, pp. 4–20, 2012.

[7] R. Rodrigues and B. Liskov, "High availability in DHTs: erasure coding vs. replication," in *Proceedings of the 4th International Conference on Peer-to-Peer Systems (IPTPS '05)*, pp. 226–239, Ithaca, NY, USA, 2005.

[8] J. Li and B. Li, "Erasure coding for cloud storage systems: a survey," *Tsinghua Science and Technology*, vol. 18, no. 3, pp. 259–272, 2013.

[9] Y. Ma, T. Nandagopal, K. P. N. Puttaswamy, and S. Banerjee, "An ensemble of replication and erasure codes for cloud file systems," in *Proceedings of the 32nd IEEE Conference on Computer Communications*, pp. 1276–1284, April 2013.

[10] W. Ma, H. Wu, and P. Liu, "Architecture and reliability of cloud storage system mass cloud," *Journal of Hohai University*, vol. 39, no. 3, pp. 348–354, 2011 (Chinese).

[11] H.-Y. Lin and W.-G. Tzeng, "A secure erasure code-based cloud storage system with secure data forwarding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 6, pp. 995–1003, 2012.

[12] D. Chen and Y. He, "A study on secure data storage strategy in cloud computing," *Journal of Convergence Information Technology*, vol. 5, no. 7, pp. 175–179, 2010.

[13] X. Xu, J. Zhou, and G. Yang, "Data privacy protection mechanism for cloud storage based on data partition and classification," *Journal of Computer Science*, vol. 40, no. 2, pp. 98–102, 2013 (Chinese).

[14] P. Prasad, B. Ojha, R. R. Shahi, R. Lal, A. Vaish, and U. Goel, "3 Dimensional security in cloud computing," in *Proceedings of the 3rd International Conference on Computer Research and Development (ICCRD '11)*, pp. 198–201, March 2011.

[15] K. Hwang and D. Li, "Trusted cloud computing with secure resources and data coloring," *IEEE Internet Computing*, vol. 14, no. 5, pp. 14–22, 2010.

[16] K. Yang and X. Jia, "An efficient and secure dynamic auditing protocol for data storage in cloud computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 9, pp. 1717–1726, 2013.

[17] S. K. Sood, "A combined approach to ensure data security in cloud computing," *Journal of Network and Computer Applications*, vol. 35, no. 6, pp. 1831–1838, 2012.