SSeCloud: Using secret sharing scheme to secure keys

Liang Hu*, Yang Huang, Disheng Yang, Yuzhen Zhang, Hengchang Liu
* University of Science and Technology of China, China
* lianghu320@gmail.com

Abstract. With the use of cloud storage services, one of the concerns is how to protect sensitive data securely and privately. While users enjoy the convenience of data storage provided by semi-trusted cloud storage providers, they are confronted with all kinds of risks at the same time. In this paper, we present SSeCloud, a secure cloud storage system that improves security and usability by applying secret sharing scheme to secure keys. The system encrypts uploading files on the client side and splits encrypted keys into three shares. Each of them is respectively stored by users, cloud storage providers and the alternative third trusted party. Any two of the parties can reconstruct keys. Evaluation results of prototype system show that SSeCloud provides high security without too much performance penalty.

1 Introduction
With the advent of cloud computing, more and more people are willing to store their personal data on cloud. Cloud storage becomes increasingly popular because it is relatively cheaper and easier to manage data. However, uncontrolled sensitive data leakage has become a major threat [1]. Once users upload their files on cloud, they lose control of their data and transfer control to semi-trusted cloud storage providers, bringing up all kinds of issues of data security: data position, data separation, data revival, data integrity protection and data audit [2].

When Cloud Storage Providers (CSP) take control of the data there are two models from encryption point of view that can help to protect the security of data stored in the cloud: (1) server-side encryption; (2) client-side encryption [3]. In the first model, data is encrypted after it is uploaded to cloud together with the encrypted files. On the other hand, the second model encrypts data before it is uploaded. However, vendors hold the encrypted key which means data is transparent for them. The evil employees working in this company may steal data combined with keys and obtain the original data resulted in insider attacks [4].

According to cloud security 2016 spotlight report, only 42% organization do not allow employees to access private cloud storage services from the corporate network [5]. Actually, these CSPs like Google Drive, Box, Dropbox and Microsoft OneDrive terms of service give their employees permission to view the files stored on their servers for some purposes [6]. Even worse many Cloud Storage Providers do not encrypt files and store data in plain text. Once CSPs suffered from adversary attacks, all data stored in the cloud would expose under the hacker especially the sensitive data [7].

In order to prevent cloud storage providers retrieving plaintext data, there is a method called zero knowledge encryption which cloud storage providers cannot decrypt data and encrypted keys stored by users. Although it guarantees CSPs never know the keys of file, users are not sure to store keys permanently. If the keys lost, and encrypted files never get decrypted.
In Depsky, using multi-cloud architecture to store data on diverse clouds to secure storage formed cloud-of-clouds [8]. In real scenario, users are not willing to store their data on several clouds. While users prefer to store less sensitive data on frequently used cloud. It is easy for users to use cloud storage, not to select four or five clouds to store data.

To address these issues, we propose SSeCloud, a novel and practical secure cloud storage system that applies the secret sharing scheme on the encrypted keys to store three shares of keys into user local disk, cloud storage system and the third trusted party. Any two parties can recover keys by secret sharing. In general, only users are more negligent to lose their key. For sharing files, users only need to share the key stored in the third trusted party combined with vendor’s key which significantly improves usability. Our contributions are mainly the following:

- We implement a secure cloud storage system SSeCloud, which all operations of encryption and decryption of data are carried out at the client side so that cloud storage provider cannot get plaintext files to snoop on users’ data. The server side only stores encrypted file and one share of encryption key. But it cannot decrypt the encrypted files with only one share of key.
- The system splits encryption keys to three shares held by user, vendor and the alternative third trusted party, which protects users’ data security and prevents key missing. Any two shares of keys can recover complete keys. In general, cloud storage provider does not collude with the third trusted party selected by users to crack the encrypted keys.
- Sharing files in SSeCloud only need to share the key stored in the third trusted party with user’s authorization. Files shared among users are always encrypted in the cloud.

2 System Design

In this section, we first present the framework design of our proposed secure cloud storage system SSeCloud. The goals of SSeCloud are as follow:

1. (1) encrypt users’ files prior to upload to protect data security;
2. (2) split keys by secret sharing scheme and simultaneously distribute to user, cloud storage provider and the third trusted party (usually we use another reliable cloud storage provider) to guarantee security;
3. (3) store data using erasure coding to save storage cost without sacrificing the reliability.

Then we briefly introduce related technology used in this system, describe the overall architecture and the features of our system (Figure 1).

2.1 Erasure Coding.
In distributed storage system, many of them use Reed-Solomon coding for storing data reliably [9]. A file to be stored is split into equal-sized blocks. There are k blocks and r additional parities for any k
shards out of \((k + r)\) are sufficient to recover the original file. And familiar with RAID technology, a strip is composed of multiple data blocks, divided into data blocks and parity blocks.

Suppose that we want to transmit a sequence of numbers \(b_0, b_1, \ldots, b_{d-1}\). What’s more, we will also assume that these numbers are in \(\text{GF}(p)\), here arithmetic is modulo \(p\), we want to reduce everything to bits, bytes, and words, so we then discuss how to compute over fields more conducive to this setting, namely fields of the form \(\text{GF}(2^r)\).

Our encoding will be a longer sequence of numbers \(e_0, e_1, \ldots, e_{d-1}\), \((p > n)\). Let
\[
P(x) = b_0 + b_1x + b_2x^2 + \ldots + b_{d-1}x^{d-1}.
\]

The polynomial \(P(x)\) can be found by using a technique known as Lagrange interpolation. Given \(d\) points \((a_0, b_0), \ldots, (a_{d-1}, b_{d-1})\), it is easy to shown that
\[
P(x) = \sum_{j=0}^{d-1} b_j \prod_{k \neq j} \frac{x - a_k}{a_j - a_k}
\]
is a polynomial of \(d\)-1 that passes through the points.

2.2 Secret Sharing.

The Shamir’s threshold secret sharing scheme is used in this system, which is based on polynomial interpolation \([10]\). The essential of Shamir’s threshold scheme is that it takes \(k\) points to define a polynomial of degree \(k-1\). This scheme is called \((k, n)\) threshold scheme, which means that a secret \(S\) can be shared by \(n\) number of users and can be reconstructed as well, if the number of shares to reconstruct exceeds some threshold value \(k\). Suppose we want to use a \((k, n)\) threshold scheme to share our secret \(S\), without loss of generality assumed to be an element in a finite field \(F\) of size \(P\) where \(0 < k \leq n < p\); \(S < P\) and \(P\) is a prime number. Choose at random \(k-1\) positive integers \(a_1, \ldots, a_{k-1}\), with \(a_i < p\), and let \(a_0 = S\). The polynomial can be built as
\[
f(x) = d_0 + d_1x + d_2x^2 + \ldots + d_{k-1}x^{k-1}.
\]

Given any \(k\) shares \(<x_i, f(x_i)>\), \(0 \leq i \leq k-1\), the secret \(S\) can be reconstructed using Lagrange Interpolation formula as follows:
\[
\sum_{i=0}^{k-1} \left( \prod_{j \neq i} \frac{x_j}{x_j - x_i} \right) f(x_i)
\]

Applying secret sharing scheme in SSeCloud, we use \((2, 3)\) threshold scheme in this scenario. An encrypted key is shared by 3 number of parties, user, cloud storage system and another trusted party (Figure 2). And the key can be reconstructing by combining with any two key shares. In this system, each party holds only one of key shares, which means any party cannot use their one key shares to decrypt files. But the user, who takes control of the files, has an absolutely ownership to the encrypted
keys and can obtain three party keys. As users can select arbitrary trusted party by themselves, cloud storage provider never have chance to get the key stored by the third trusted party.

In our system, there provide basic operation including upload and download file, we will describe these two functions in details in next section. Here we introduce the two core features to improve security:

2.3 Retrieve key.
One of three holders lost key, using secret sharing can recover key. Actually only users are more negligent to forget their key. Suppose in this situation, our retrieve key operation divides into two processes. Consider of security, cloud storage provider hold one share of keys all the time at most. Key reconstruct and split operations are doing at client side.

In the first stage, users select to download keys held by cloud storage provider and the third trusted party to client side. Then compose to a complete key by secret sharing. At this time, a full key still controlled by users. Cloud storage provider and the third trusted party never retrieve a complete key to some files.

Secondly, dividing key to three shares apply secret sharing again and storing user's share key to local disk to finish retrieving key operation. What’s more, vendor and another cloud storage provider don’t collude with each other to crack the key in general.

2.4 Sharing file.
In SSeCloud, if users want to share their files, only need to share the key stored in the third trusted party. According to the shared key, the one who is shared can find the file stored in cloud storage system distributor used. As long as being shared one is trusted, the shared file is secure relatively.

3 Algorithm Design and Analysis

3.1. Upload file
**Description:** First, at the client side encrypt file \( f' \) and encode \( f' \) to n data blocks using Reed-Solomon erasure coding stored in cloud storage C; Simultaneously, Encryption keys divided into 3 shares, respectively saved in user local disk L, cloud storage system C and another trust one T through multi-threading.

| Algorithm 1: Algorithm of upload file operation |
|-----------------------------------------------|
| **Data:** Plaintext file \( f \) |
| **Result:** Store file and keys |
| **for** (each file: \( f_i \)) **do** |
| \( <f_i, Key> = encrypt(f) \); |
| **// parallel execution** |
| **Thread 1:** // encoding file and store data blocks |
| Blocks \( n \) (\( k \) shards \( + r \) parities) = RS erasure encoding (\( f' \)); |
| **for** (each block: \( n_i \)) **do** |
| hash and store each block \( n_i \) into \( C \); |
| **end** |
| **Thread 2:** // split keys and store key shares |
| Shares \( N_i \) (\( i = 1,2,3 \)) = secret sharing split (\( Key \)); |
| **for** (each share: \( N_i \)) **do** |
| store \( N_i \) into \( L \); |
| store \( N_2 \) into \( C \); |
| store \( N_3 \) into \( T \); |
| **end** |
| **end** |
3.2. Download file

**Description:** When user requests to download a file f, SSeCloud system construct the encrypted file f' at the server side to find k shards belong to f'; check hash value of each shards if it is tampered; then use Reed-Solomon code to decode k shards recovering encrypted file f'. Simultaneously, usually we use local key and SSeCloud key to reconstruct Key using secret sharing at the client side. Only when users lose their local key, then use the third trusted party key to instead of local key. Lastly, combine encrypted file with encrypted key to get original file f.

**Algorithm 2:** Algorithm of download file operation

| Data:     | Data blocks n and Encryption keys 2 shares |
|-----------|-------------------------------------------|
| Result:   | Plaintext original file                   |
| Request downloading f;  |
| // parallel execution |
| **Thread 1:** // construct encrypted file at server side     |
| Find k shards among n blocks of f';  |
| for (k shards) do    |
| check shards hash to avoid data tampering; |
| end    |
| get f'; use RS erasure coding to decode k shards;    |
| downloading f' ;    |
| **Thread 2:** // generate Key at client side       |
| Search key share N2 of f' at C;                |
| Download N2 to client;                          |
| Key = secret sharing reconstruct (N1, N2);        |
| f = decrypt (f', Key) ;                          |

4 Implementation and Evaluation

4.1 Implementation.

We have implemented a SSeCloud prototype system that supports the upload, download, share file and retrieve key operations. The Reed Solomon erasure coding and Shamir secret sharing algorithm are implemented by Java. Files are split to 6 shards for 2 parity shards.

The encryption is performed at client side with AES 256-bit symmetric key [11], SHA-256 for cryptographic hashes [12]. The cloud storage system uses Hadoop Distributed File System (HDFS [13]) and sets 64M block size, 1 replication because we have used erasure coding for redundancy already. The third trusted party we use a server instead for emulation.

| Test Content | Test Result |
|--------------|-------------|
| Confidentiality: Directly check the data stored with the key share in the HDFS. | Both data chunks and complete encrypted file are messy code. One share can’t decrypt file. |
| Integrity: Directly tamper data chunk in HDFS. | Check hash for errors. |
| Availability: Delete two data chunks of a file. | This file still can be downloaded. |
| Retrieve Key: Suppose user lost key. | Get key back by regenerating key. |

4.2 Evaluation.

We mainly from the two points to evaluate our system, security and performance, when applying the secret sharing scheme to secure keys. The security evaluations are list in Table 1 in details. We test
data confidentiality, integrity, availability and retrieve key. It shows that all aspects achieved our goals of this system.

As for performance test, we calculate the average uploading and downloading time for files size 32MB, 64MB, 128MB, 256 MB for ten times respectively, from two aspects including encrypt files operation and unencrypt (Figure 3 and Figure 4). There’s no doubt encrypt operation will cause overhead for improve security.

![Figure 3: The average uploading time](image)

![Figure 4: The average downloading time](image)

But we can see that with files larger, the overhead will be less. In average uploading time, the rates of unencrypted to encrypted are 59.5% in 32MB file size, 75.7% in 64MB file size, 86% in 128MB file size and 95% in 256MB file size, while in average downloading time they are 53.4%, 77.3%, 87.3% and 94.9%.

However, although there’s a wide gap between small files, the total time of uploading and downloading time is in several seconds which users can bear it completely. In fact, for large files transmission time takes over a majority of overhead.

### 5 Conclusion

In this paper, we present a secure cloud storage system, SSeCloud, for managing keys via secret sharing scheme. SSeCloud provides data security on the client side encrypting all the time. Besides, recovering key to prevent missing. Users store three key shares separately into local, cloud storage provider and the third trusted party. Normally the third party will not be involved. Only when users lose their local key can they look for the third party. CSPs never have chance to get other two key shares as the third party selected by user discretionarily. Our evaluation shows that SSeCloud has high security of data confidentiality, integrity and availability. Due to the addition of security mechanism,
SSeCloud has an extra performance overhead about 5% in large file, and it is within the user’s tolerance.

Acknowledgements
In this paper, the research was financially sponsored by the National Natural Science Foundation of China (Project No. 61472384).

References
[1] Vijayalakshmi V, Rohini T, Sujatha S, et al. “Survey on detecting leakage of sensitive data[C]”, Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave), World Conference on. IEEE, pp. 1-3, (2016).
[2] Dr.P.K. Rai, R.K. Bunkar, Vivekananda Mishra. “Data Security and Privacy Protection Issues in Cloud Computing [J]”, IOSR Journal of Computer Engineering, 16 (1), 39-44, (2014).
[3] Wilson D C, Ateniese G. “To share or not to share” in client-side encrypted clouds[C],” International Conference on Information Security. Springer International Publishing, pp. 401-412, (2014).
[4] Rocha F, Correia M. “Lucy in the sky without diamonds: Stealing confidential data in the cloud[C]. Dependable Systems and Networks Workshops (DSN-W)”, 2011 IEEE/IFIP 41st International Conference on. IEEE, pp. 129-134, (2011).
[5] Cloud Passage. Cloud Security 2016 Spotlight Report. 2016. https://pages.cloudpassage.com/rs/857-FXQ-213/images/cloud-security-survey-report-2016.pdf
[6] Sync. “Sync: White Paper.”, September 2015. https://www.sync.com/pdf/sync-privacy.pdf
[7] Pasquale L, Hanvey S, Mcgloin M, et al. “Adaptive evidence collection in the cloud using attack scenarios[J],” Computers & Security, 59, pp. 236-254, (2016).
[8] Bessani A, Correia M, Quaresma B, et al. “DepSky: dependable and secure storage in a cloud-of-clouds[J],” ACM Transactions on Storage (TOS), 9(4), pp. 12, (2013).
[9] Alexandros G. Dimakis, P. Brighten Godfrey, Yunnan Wu, Martin Wainwright and Kannan Ramchandran. “Network coding for distributed storage systems”, IEEE Transactions on Information Theory, Volume: 56, Issue: 9, Sept. (2010).
[10] Shamir A. “How to share a secret [J],” Communications of the ACM, 22(11), 612-613, (1979).
[11] Biryukov A, Khovratovich D, Nikolić I. “Distinguisher and related-key attack on the full AES-256[M],” Advances in Cryptology-CRYPTO 2009. Springer Berlin Heidelberg, pp. 231-249, (2009).
[12] Mendel F, Pramstaller N, Rechberger C, et al. “Analysis of step-reduced SHA-256[C]”, International Workshop on Fast Software Encryption. Springer Berlin Heidelberg, pp. 126-143, (2006).
[13] Rashmi K V, Chowdhury M, Kosaian J, et al. “EC-cache: load-balanced, low-latency cluster caching with online erasure coding[C],” 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). USENIX Association, pp. 401-417, (2016).