ISBN 978-981-11-3671-9
Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering (WCSE 2017)
Beijing, 25-27 June, 2017, pp. 784-788
doi: 10.18178/wcse.2017.06.137

Cloud Security Gateway Based on Real-Time Processing

Jingsi Ren [1,2+], Jinlin Wang [1], Xiao Chen [1] and Xiaozhou Ye [1]

[1] National Network New Media Engineering Research Center, Institute of Acoustics, Chinese Academy of Sciences, Beijing 100190, China

[2] University of Chinese Academy of Sciences, Beijing 100049, China

Abstract. Cloud storage offers a flexible, dynamic and cost effective data storage service. However, there are some major concerns regarding the security and privacy of data in cloud storage. This paper presents design and implementation of a Cloud Storage Security Gateway (CSSG) based on real-time processing. The CSSG serves all clients in an organization that outsources its data through HTTP based interface protocol. The outsourced data is encrypted on uploading and decrypted on downloading in real-time on the proposed CSSG. The session between the client and the Cloud Service Provider (CSP) is not interrupted. Hence the data confidentiality during communication and storage is effectively safeguarded. In addition, the CSSG remains transparent to both the client and CSP, therefore it can be used seamlessly with existing applications. The experimental results validate the efficiency of the proposed CSSG.

Keywords: cloud storage, data confidentiality, security gateway, real-time processing

1. Introduction

Cloud computing [1] has emerged as the next-generation architecture of enterprise and individual IT users. Cloud storage [2] service, which is an important branch of cloud computing, allows clients to store large data on the Cloud Service Provider (CSP). Such outsourcing of data storage enables clients to focus on innovations and get relief from the burden of constant server updates and other computing issues [3]. However, the fact that data owners no longer physically possess their sensitive data raises new challenges to data confidentiality in cloud storage. The sensitive data is often encrypted before outsourcing to remote servers for confidentiality. The encryption is performed either at the client-side or the server-side. In this paper, we propose a novel Cloud Storage Security Gateway (CSSG) based on real-time processing, which can encrypt/decrypt data file without interruption of session between the client and the CSP.

2. Related Work

In the aspect of storage security, the mainstream cloud storage systems can be divided into three categories based on the security mechanism [4]:

The first category of cloud storage systems such as iDisk [5], does not encrypt user data files. The plaintext data is stored directly on the CSP.

The second category of cloud storage systems applies server-side encryption on user data files before they are stored in the server. The CSP keeps the encryption keys. This category includes Drop Box [6], AWS S3 [7], et. al.

In the third category of cloud storage systems, the users are responsible for encrypting the data files before they are uploaded to CSP. The encryption keys are managed in a hierarchical manner, in which the

+ Corresponding author. Tel.: + 15654398887; fax: +010-82547890.
E-mail address: yexz@dsp.ac.cn.

users themselves hold the root key while other keys are stored at the cloud server as ciphertext. This category includes SpiderOak [8], Wuala [9] etc

These three security mechanism reflect the tradeoffs between data confidentiality and performance. The first category of cloud storage system is the least secure. Anyone with access to the underlying storage medium can view user data as plaintext. In the second category of cloud storage systems, the CSP holds user data in ciphertext and keeps the encryption key. However, there are still potential threats to data confidentiality, as the cloud storage server may be vulnerable to malicious attacks resulting in loss of data. The third category of cloud storage systems provides the highest level of security among all. However, it can be the least efficient in terms of transmission delay and client computational burden, especially when a large amount of data needs to be encrypted and decrypted.

In order to consider both the data confidentiality and performance, we propose a novel cloud storage security gateway. In the CSSG, user data files are encrypted when uploaded and decrypted when downloaded. The session between the client and the CSP is not interrupted thus the CSSG is transparent to both ends.

3. Proposed Cloud Storage Security Gateway

3.1. Working mechanism

The proposed CSSG is designed for cloud storage systems with HTTP application programming interface (API). In the standard process, the client initiates a login request after establishing a TCP connection with the cloud server. The cloud server verifies the user name and password, and then responds with an authorization token. At this moment, the client can issue a file upload request. The HTTP request message body contains the contents of the file to be uploaded. When the entire file has been uploaded, the cloud server will calculate the MD5 value of the file and attach it to the HTTP response message. Similarly, when the client downloads a file, the client issues the download request. The cloud server returns the download response after finding the corresponding file and MD5 value. The file is transmitted in the HTTP response message body. This entire interaction process can be divided into signaling stream and data stream based on the function. The signaling stream includes user login request, authorization token, file upload request and file download request. The data stream contains uploaded file and downloaded file.

In this paper, we introduce a CSSG between the client and cloud server. The CSSG establishes TCP connections with both ends and transfers the signaling stream without any modification, thus the original interactive logic does not change. However for the data, the CSSG encrypts the upstream and decrypts the downstream with symmetrical cryptography such as AES. It is worth noting that the session between client and cloud server is not interrupted by CSSG.

3.2. System implementation

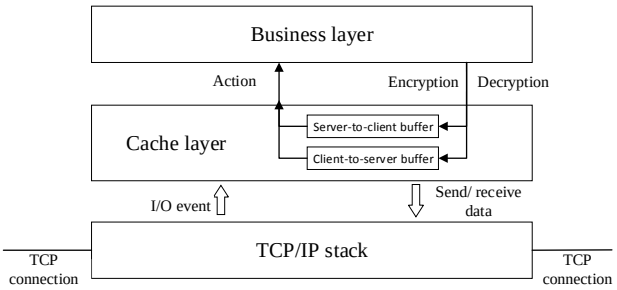

Fig. 1: System implementation

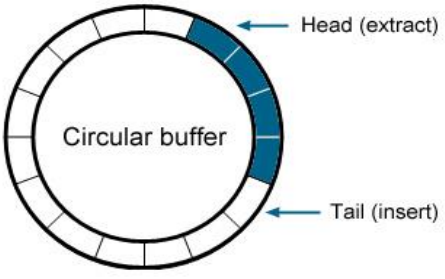

Fig. 2: Circular buffer

The CSSG is designed as a hierarchical structure. As shown in Fig. 1, there are three layers in the proposed system: 1) the TCP/IP protocol stack; 2) the cache layer; and 3) the business layer. The TCP/IP protocol stack maintains two separate TCP connections with the client and the cloud server. These two TCP connections make a pair and they are managed together. When a socket interface reads or writes data, corresponding socket IO events will drive the cache layer. The cache layer is the key to splice two TCP connections and actuate the whole system. The data flows in two directions: from client to cloud server, and from cloud server to client. Therefore, there are two buffers for a pair of TCP connections: Client to Server

Event-Driven Circular Buffer (CS-EDCB), and Server to Client Event-Driven Circular Buffer (SC-EDCB). Each of these two buffers caches data from one end, and triggers actions in business layer and TCP/IP stack. The business layer is responsible for the service logic such as user attribute management, data integrity verification, HTTP parsing and data encryption/decryption.

3.3. Cache layer

The CSSG splices TCP connections from the client-side and the cloud server. These two distinct TCP connections have different transmission bandwidth, delay, and jitter based on their capacity and network status. In this scenario, the core task of the cache layer is to coordinate the transmission rate at both ends, and make seamless conjunction such that the CSSG is transparent to both ends. Thus, an event-driven circular buffer is proposed to manage a pair of TCP connections on one direction. As shown in Fig. 2, a circular buffer is a memory allocation scheme where memory is reclaimed when an index (incremented modulo the buffer size) writes over a previously used location. We extend the standard circular buffer to event-driven circular buffer in such a way that it is driven by network IO events and triggers corresponding actions to process the data flow. There are four network IO events in the proposed design:

- Client-socket-can-read (E-CSCR): there are enough data in client socket receive buffer to read.
- Client-socket-can-write (E-CSCW): there are enough empty space in client socket send buffer to copy data to.
- Server-socket-can-read (E-SSCR): there are enough data in server socket receive buffer to read.
- Server-socket-can-write (E-SSCW): there are enough empty space in server socket send buffer to copy data to.

In Linux operating system, we can use kernel system call "epoll" to monitor multiple socket file descriptors and get I/O events notification.

The corresponding action for each event can be described as four algorithms.

Algorithm 1 Action for E-CSCR

```
//Get the length of empty space in circular buffer
Len ← (Head − Tail) mod Size
if Len > 0 then
    Receive Len bytes data from client socket
    //Update the index in circular buffer
    Tail ← (Tail + Len) mod Size
    Encrypt data stream
end if
```

Algorithm 2 Action for E-CSCW

```
//Get the length of data in circular buffer
Len ← (Tail − Head) mod Size
if Len > 0 then
    Send Len bytes to client socket
    //Update the index in circular buffer
    Head ← (Head + Len) mod Size
end if
```

Algorithm 3 Action for E-SSCR

```
//Get the length of empty space in circular buffer
Len ← (Head − Tail) mod Size
if Len > 0 then
    Receive Len bytes data from server socket
    //Update the index in circular buffer
    Tail ← (Tail + Len) mod Size
    Decrypt data stream
end if
```

Algorithm 4 Action for E-SSCW

```
//Get the length of data in circular buffer
Len ← (Tail − Head) mod Size
if Len > 0 then
    Send Len bytes to server socket
    //Update the index in circular buffer
    Head ← (Head + Len) mod Size
end if
```

A session between client and cloud server includes two event-driven circular buffers: CS-EDCB and SC-EDCB. Each of the two buffers monitors distinct network I/O events and triggers corresponding actions as illustrated in Table 1

Table 1: Event-driven circular buffer

Buffer	Event	Action
CS-EDCB	E-CSCR	Algorithm 1
	E-SSCW	Algorithm 4
SC-EDCB	E-SSCR	Algorithm 3
	E-CSCW	Algorithm 2

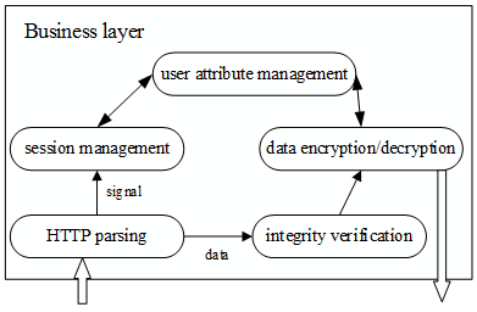

Fig. 3: Business layer structure

3.4. Business layer

As shown in Fig. 3, the business layer includes HTTP parsing module, session management module, integrity verification module, user attribute management module, and data encryption/decryption module.

The HTTP parsing module is the first to receive data from cache layer. All the data flowing through CSSG is based on HTTP protocol and the function of this module is to parsing these flows into signaling stream and data stream. In the signaling stream, HTTP verbs are analyzed to determine user actions such as file uploading (PUT/POST), file downloading (GET), and file deletion (DELETE). User login information and authorization token is also parsed in this module. The data stream is then forwarded to integrity verification module for further processing.

The session management module is responsible for managing the session interaction state between client and cloud server. A state machine is designed to keep track of session states with following defined states: New_Session, Login_Request, Logged_In, File_Uploading, File_Downloading and Delete_Session. The state machine transfers between these states based on the signaling of HTTP parsing module.

The user attribute management module is responsible for generating, managing and saving user information. The user information includes the user list, encryption key, file directory tree and the access log. For security and audit needs, this user information is centrally managed and regularly backed up.

The integrity verification module is used to check the integrity of both plaintext file from client and encrypted file from cloud server. The client calculates MD5 value of the data as M1 before uploading. When the cloud server receives the uploaded file, it replies with the MD5 value of the received file (M2). In order to check the integrity of both the files, the integrity verification module calculates the MD5 value of data file received from client as M3 and the MD5 value of data file forwarded to cloud server as M4 in the uploading process. After receiving the response from cloud server, the integrity verification module checks whether M4 equals to M2. The received encrypted file is considered intact if the integrity test at CSSG is successful. In that case, the integrity verification module will replace M2 with M3 in the response to client. Similarly, the client can check the integrity of uploaded file by verifying whether M1 equals M3.

The data encryption/decryption module is responsible for encrypting upstream data and decrypting downstream data with user encryption key. As the TCP payload is of variable length, we adopt the stream cipher such that variable-length incoming data from TCP/IP stack can be processed and forwarded immediately. A stream cipher is a symmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream (keystream). In a stream cipher, each plaintext digit is encrypted one at a time with the corresponding digit of keystream to generate a digit of the cipher text stream. As a result, the ciphertext and plaintext remain the same length that helps to keep CSSG transparent in the process.

4. Experimental Test

In order to verify the feasibility and efficiency of the proposed CSSG, CSSG was compared with the traditional client encryption method, where the data files were encrypted by the client terminal before uploading to cloud server and decrypted after downloading. The client, cloud server and CSSG were executed on three Linux servers with Intel Xeon E5-2620 2.00GHz CPU and 16 GB memory. Both the client and cloud server deployed OpenStack Swift cloud storage system and they were connected directly using a Gigabit Ethernet connection. In the case of CSSG, the client and cloud server were connected through the CSSG using same Ethernet connection. All the programs were written in C++ using the OpenSSL 1.0.1 library. Both schemes used the same 256-bit AES-CTR encryption algorithm.

4.1. File upload test

The size of test file varied from 20MB to 200MB. In the client encryption scheme, the client terminal encrypted the test file and then uploaded to cloud server. While in the CSSG scheme, the test file was uploaded from client to cloud server passing through CSSG. The timing results are shown in Fig. 4.

4.2. File download test

Similar to the upload test, the size of test file in download test varied from 20MB to 200MB. In the client

encryption scheme, the client terminal downloaded and decrypted the test file. While in the CSSG scheme, the test file was downloaded from cloud server to client passing through CSSG. The timing results are shown in Fig. 5.

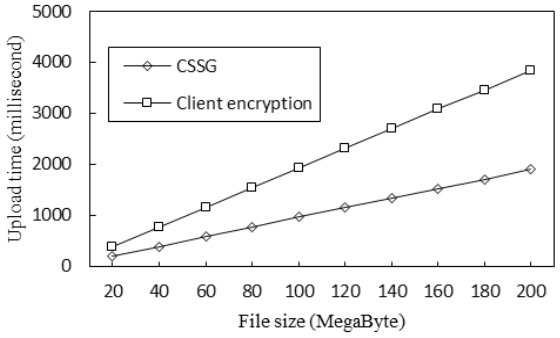

Fig. 4: Test results of file upload

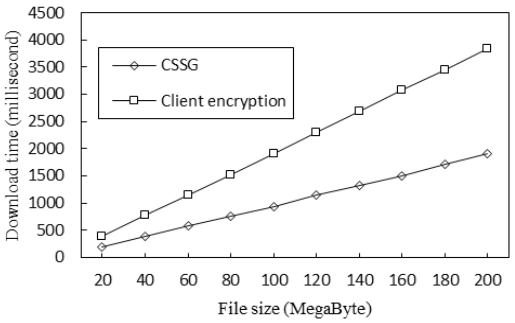

Fig. 5: Test results of file download

4.3. Performance analysis

As seen in the test results, the time spent in uploading, downloading, encryption and decryption increases significantly with the size of data file. In the experimental environment, the file encryption/decryption time and transmission time are roughly same. When CSSG is used in the system, the encryption and upload time reduces by 50%, and similar figures can be observed in the file decryption and downloading process. The results clearly indicate that the burden of data encryption and decryption is effectively unloaded by CSSG.

5. Conclusions

In this paper, we propose a novel cloud storage security gateway that can process the interactive data streams in real time. The CSSG is implemented in a hierarchical structure and deployed as a security gateway to serve all the clients in an organization. The CSSG ensures confidentiality and integrity of outsourced data and improves the efficiency of data upload and download processes. However, current design does not consider some important security features such as Provable Data Possession and Fine-Grained Access Control. These security features will be included in the future implementation of the proposed gateway.

This work has been supported by Pilot Program of Chinese Academy of Sciences (XDA06010302) and Pioneer Program of Institute of Acoustics, Chinese Academy of Science (Y654101601).

6. References

[1] J. W. Rittinghouse and J. F. Ransome. *Cloud computing: implementation, management, and security*. CRC press, 2016.

[2] W. Qi and J. Wang. A Framework of Secure Cloud Storage in the Age of Big Data. *Journal of Network New Media*. 2016, 5 (2): 1-7.

[3] T. Li, L. Hu, Y. Li, J. Chu, H. Li and H. Han. The Research and Prospect of Secure Data Access Control in Cloud Storage Environment. *Journal of Communications*. 2015, 10 (10): 753-759.

[4] Y. Fu, S. Luo and J. Shu. Secure online storage system based on cloud storage environment. *Journal of Software*. 2014, 25 (8): 1831-1843.

[5] https://en.wikipedia.org/wiki/IDisk [2016-12-29]

[6] https://www.dropbox.com/business [2016-12-29]

[7] https://aws.amazon.com [2016-12-29]

[8] https://spideroak.com [2016-12-29]

[9] https://wuala.com [2016-12-29]