Research on Multi-user Data Sharing Service Based on Object Storage

Huiran Zhang[1, 2, 3], Xin Wang[1], Shengzhou Li[1], Tao Xu[1], Zhiting Guo[1], Hongqing Hu[1], Xue Chen[1], Chengfan Li[1], Xiao Wei[1], Quan Qian[1] and Dongbo Dai[1, a]

[1]School Computer Engineering and Science, Shanghai University, Shanghai 200444, China

[2]Shanghai Institute for Advanced Communication and Data Science, Shanghai University, Shanghai 200444, China

[3]Materials Genome Institute of Shanghai University, Shanghai University, Shanghai 200444, China

[a]dbdai@shu.edu.cn

Keywords: Cloud storage, Data sharing, Data security, Access control.

Abstract: Existing Object Storage providers generally use signature as data file access credential, and AWS Signature method is widely recognized as a standard security notion. However, most of the Object Storage service suffer from the disadvantages of weak data sharing security, which has severely impeded the daily usages of cloud storage users. In this paper, we propose a new framework, which called Multi-user Data Sharing Scheme (MDSS), bases on Object Storage for the security of sensitive personal data in data sharing. For the sake of data security, permission authentication algorithm is added to AWS Signature method. In addition, this scheme can solve the problem of real-time synchronization of multi-machines data and fine-grained data sharing strategy creation.

1. Introduction

With the popularity of cloud computing, people are gradually getting accustomed to a new method of data sharing in which the data is stored on the cloud and the hosts are used to manipulate data from the cloud [23]. Personal host only have limited storage spaces and computing performance. On the contrary, the cloud has massive number of resources. As a result, to address the problem of limited resources, using cloud resources is a feasible solution [12].

There are three main solutions used for storage in the cloud: Block storage, File storage and Object storage. The Block storage has two types of Direct- Attached Storage (DAS) [25] and Storage Area Network (SAN) [18]. Block storage has the advantages of high input/output (I/O), low latency, and high reliability. Many cloud computing services such as AWS elastic block storage [8], Azure premium storage [14], Google persistent disks [20] and Cinder block storage services for OpenStack [11], provide block storage solution for personal host. But it also has some in-surmountable defects like poor expansibility, inability to provide cross-machine data migration services [7]. File storage provides data storage services through accessing file. Network Attached Storage (NAS) [6], as a common file storage, can provide file sharing and data backup functions for different operating systems. Object storage is fundamentally different from traditional block or file storage systems [3]. Object storage organizes information into containers in flexible sizes, referred to as objects. Object storage has the advantages of low latency of block storage and sharing of file storage. The Object storage examples are AWS S3 [16] [23], Swift object storage services for OpenStack [1], etc.

Data sharing is a very important practical requirement for users. Nowadays, cloud storage services are widely used, and the data security problem becomes more and more severe [13]. User can upload data to the cloud and share their data with other users. These three storage methods can solve the problem of data recovery and limited storage space. But some of the above methods are for single-user services, which have not data sharing feature, such as Block storage. Although File storage and object storage provide data sharing, but it is not perfect for sharing permissions. These methods are not meet all the requirements of data owners. Data privacy of sensitive personal data is a great concern for data owners. In this case, data privilege management is a major issue.

Apparently, the cloud storage service provider should handle not only data recovery and limited storage space, but also the security of sensitive personal data. Rawal et al. [19] propose a secure disintegration protocol (SDP) for the protection of privacy on-site and in the cloud. Ning et al. [17] propose the first accountable authority revocable CP-ABE based cloud storage system with white-box traceability and auditing, referred to as CryptCloud+. However, both SDP and CryptCloud+ aim to solve the security problem of user data in the cloud, and do not focus on the security problem of user private data sharing. Private data sharing is an urgently-to-be-solved problem for multiple users. When the same dataset is applied to different studies, or various dataset is used for the same research, the copy method is not an efficient sharing strategy if the amount of data is large. Furthermore, if data owner wants to share data with time limit, host limit and other limits, the problem will become more complicated. In this paper, we propose a new framework, which called Multi-user Data Sharing Scheme (MDSS), bases on object storage for the security of sensitive personal data. This scheme can solve the problem of real-time synchronization of multi-machines data and fine-grained data sharing strategy creation.

2. Our Proposed Mechanism

2.1. System Architecture.

As shown in Fig. 1, the architecture of MDSS in cloud consists of four entities DO (Original Data Owner), DU (Data User), TA (Trusted Authority) and OSP (Object Storage Provider).

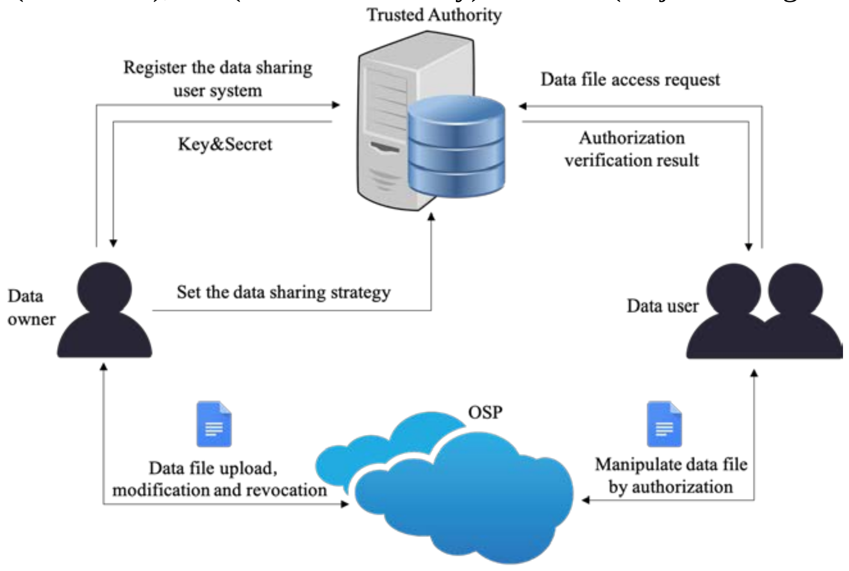

Trusted Authority
Register the data sharing user system
Key&Secret
Data file access request
Authorization verification result
Set the data sharing strategy
Data owner
Data user
OSP
Data file upload, modification and revocation
Manipulate data file by authorization

Fig. 1 The architecture of MDSS

(1) DO is an entity with original data who wishes to store data on cloud storage server maintained by OSP and share the specified data securely with others. DO has the highest authority of data operation such as share, delete, update, and so on. Before DO uploads data to OSP, DO registers the data sharing user system in TA to get Key&Secret information. Key&Secret is used to verify the user's identity and permissions to manipulate data. DO must use Key&Secret to operate cloud data including upload original data.

(2) DU is an entity who uses shared data in the OSP. The data operation permission of DU are controlled by DO, including operating permissions, time limits, resharing limits, host limits, and so on. In fact, DU first sends data file access request to TA, and then TA validates the user's request and returns authorization verification result. Finally, the DU manipulate data file in the OSP requires authorization verification results as credentials.

(3) TA is an entirely credible entity who is responsible for generating and distributing Key&Secret for members in the data sharing user system. DO can share specified data to members and set the data sharing strategy. DU requires to use Key&Secret to validate the operation, which is divided into user identity authentication and data operation authorization authentication. Once the identity and operation are verified, TA will return authorization verification results to DU. In addition, in the process of data sharing, DO can change the data sharing strategy at any time.

(4) OSP is a semi-trusted entity who is in charge of saving data file of DO. It will execute the user data file access request with credentials.

Our proposed scheme can be divided into five parts: user registration, original data file upload, data sharing strategy creation, authorization verification and data file access.

(5) User registration: DO and DU are user in MDSS. A user can join the multi-user data sharing system by committing a registration request to TA, who issues a Key&Secret to the user.

(6) Original data file upload: DO use Key&Secret as credential to upload original data file into CSP for saving and sharing.

(7) Data sharing strategy creation: DU establishes the specified data sharing strategy by sending a share request to TA. The strategy includes time limit, data operation limit, host limit, mount times limit, resharing limit, etc.

(8) Authorization verification: User needs to verify user identity and data operation permissions before manipulating data in CSP. TA return the validation results, which are used in the subsequent data file access by user.

(9) Data file access: User can manipulate data file in CSP by authorization.

2.2. User mode.

Most of the existing solutions are developed for the alone-user mode, and there is no way to meet the diverse needs of users, such as data sharing problems. Therefore, multi-user mode will become the inevitable trend of cloud computing development. The MDSS is a framework for addressing the inability to provide sensitive data sharing services in an alone-user mode. The MDSS can be applied flexibly in alone-user mode and multi-user mode according to different requirements.

2.3. Alone-user mode.

Alone-user mode applies to users without data sharing requirements. In this mode, users can utilize cloud resources to solve problems with local capacity. Single-user multi-bucket policy is used to separate the file data from the personal host in alone-machine mode. The data owner marks the data with a pair of Key&Secret tags, with Bucket as the minimum mount unit. The data owner can use the same pair of Key&Secret to mount a Bucket on multiple hosts for synchronize data. Even if the host crashes for no reason, user will not lose any data because data is stored in the cloud. This mode also improves the security and portability of the data.

2.4. Multi-user mode.

Multi-user mode is an extension of Alone-user mode for users with data sharing requirements. Data sharing between users is often required, but sharing data by copying is unsafe, unreliable, and inefficient. Consequently, its necessary to change Alone-user mode to Multi-user mode. In this mode, data owner can upload their photos, videos, documents and other files to the OSP and share this data with other users. In addition, data owner can set information such as operation permissions, time limits and resharing rights when sharing data to other users. The MDSS verifies the validity of the request when data user requests to operate shared data, and the OSP only responds to legitimate requests.

2.5. Design MDSS.

In this section, we present our scheme in detail. The MDSS is designed for the sensitive personal data sharing in the cloud. The main process of MDSS includes user identity authentication module, user operation authentication module, AWS S3 API compatible module and user operation API module.

2.6. User Identity Authentication.

The first step of MDSS after receiving a data request is to verify user identity legitimacy. During these years, more users choose to store data in the cloud for persistent storage [25]. Although cloud computing's benefits are tremendous, security and privacy concerns are the primary obstacles to wide adoption [20] [5]. Since MDSS is designed based on object storage, we use AWS S3 signature verification process as the basis for permission verification. In the object storage, users utilize Key&Secret to prove their identity. When requesting an operation, data user needs to add some specified request headers according to the rules, which must carry the legally calculated completed signature. The users identity is authenticated by MDSS before it responds to user action requests. The AWS Signature Version 4 process is shown as Fig. 2.

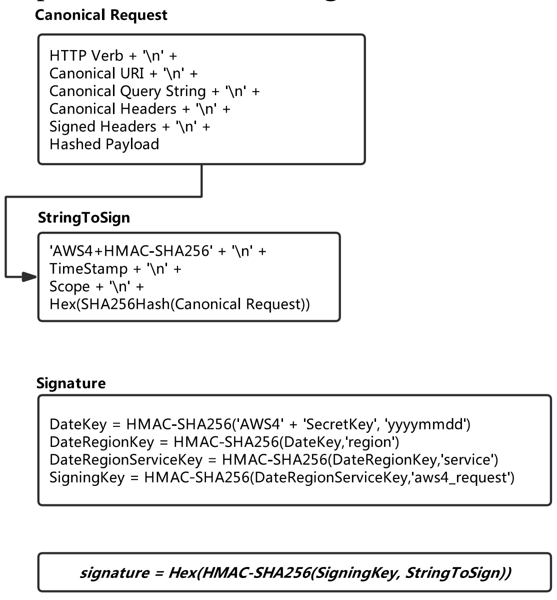

Canonical Request
HTTP Verb + '\n' +
Canonical URI + '\n' +
Canonical Query String + '\n' +
Canonical Headers + '\n' +
Signed Headers + '\n' +
Hashed Payload
StringToSign
'AWS4+HMAC-SHA256' + '\n' +
TimeStamp + '\n' +
Scope + '\n' +
Hex(SHA256Hash(Canonical Request))
Signature
DateKey = HMAC-SHA256('AWS4' + 'SecretKey', 'yyyymmdd')
DateRegionKey = HMAC-SHA256(DateKey,'region')
DateRegionServiceKey = HMAC-SHA256(DateRegionKey,'service')
SigningKey = HMAC-SHA256(DateRegionServiceKey,'aws4_request')
signature = Hex(HMAC-SHA256(SigningKey, StringToSign))

Fig. 2 AWS Signature Version 4 process

Table.1. UIA Algorithm

Algorithm 1. The user authority verification algorithm.

Input: User's operation request with AWS S3 V4 signature parameters.
Output: Operation response.
Step 1. Extract request signature RS from user's operation request.
Step 2. Extract the user's AccessKey, type of request operation, data of request operation and other relevant information from the user operation request.
Step 3. Get user's SecretKey and authority information from platform database according to AccessKey.
Step 4. Calculate server signature SS using the AWS S3 V4 computation process.
Step 5. Compare signatures RS with SS. If the two signatures are equal, the platform execute the user's operation request and returns the corresponding information. If the two signatures are different, the platform will reject the user's operation request and return an error code.

In fact, the AWS S3 storage protocol is to add some specified request headers to the request operation. Signature is used to verify the identity of the requested action, prevent data tampering and the signature from being stolen. Client uses the Key&Secret, HTTP method and other information to get the signature according to the AWS signature calculation process, and adds the signature to the request header. The MDSS verifies the identity of the request operation in line with AWS Signature Version 4. The SignedHeaders element in the request header is designed to protect data security and prevent file tampering. The x-amz-date element is to prevent the signature from being stolen. Authenticating user identity is the first issue to be addressed. We propose the UIA algorithm to verify the validity of user identity. The Algorithm is shown in the table below.

2.7. User Operation Authentication.

After the user identity authentication is successful, MDSS will perform operation permission verification, and only the operation that is verified will get the correct response. Data operation has five levels of authority in MDSS, from low to high, respectively is Get, Post, Put, Delete, Share. The data owner has the highest data operation authority. Data owner can share data to other users on demand, so data users operation permission is set by data owner. Data user operating requirement has to be less than the current authority level. The MDSS will verify the user’s operation request according to the recorded information, which can prevent data leakage and ensure data security and reliability.

2.8. AWS S3 API Compatible module.

The upper layer of the user permission authentication module is the API module, including AWS S3 API Compatible module and User Operation API module. The function of the compatible module is to realize the migration of user data by compatible with the standard API in the market, to avoid data loss caused by the user replacement of OSP. The API compatible module implements 10 major AWS S3 API operations, which includes service API, bucket API and object API. The implementation of the MDSS API refers to AWS S3 REST API introduction. To ensure that users can apply the S3fs tool to mount buckets in the storage service, the request header and response header of the platform API are consistent with the descriptions in the AWS S3 REST API documentation. The compatible API operation is shown in Table 2.

Table.2. The URL design of API compatible module

Id	URL	Method	Description
1	/	GET	This implementation of the GET operation returns a list of all buckets owned by the authenticated sender of the request.
2	/	PUT	This implementation of the PUT operation creates a new bucket.
3	/bucket	DELETE	This implementation of the DELETE operation deletes the bucket named in the URI.
4	/bucket	HEAD	This operation is useful to determine if a bucket exists and you have permission to access it.
5	/bucket	GET	This operation is useful to determine if a bucket exists and you have permission to access it.
6	/bucket/object	PUT	This implementation of the PUT operation adds an object to a bucket.
7	/bucket/object	DELETE	This implementation of the DELETE operation deletes the object named in the URI.
8	/bucket/object	POST	This implementation of the POST operation deletes the multi-object named in the URI.
9	/bucket/object	HEAD	The HEAD operation retrieves metadata from an object without returning the object itself.
10	/bucket/object	GET	This implementation of the GET operation retrieves objects from server.

2.9. User Operation API module.

This module is used for user information management, including user management, Key&Secret management, third-party user system compatibility and data sharing operations. Data owner uses the data sharing API to implement the specified data sharing operation. By default, data user only has minimal data manipulation permissions. Once the user’s Key&Secret is generated, no modifications are allowed, but the MDSS supports deletion and re-creation. Basically, AWS S3 API

Compatible module provides the service for direct operational data. User Operation API module implements access control by managing user information. The API design of this module is shown in Table 3.

Table.3. The URL design of user operation API module

Id	URL	Method	Description
1	/key	POST	This implementation of the POST operation adds a Key&Secret to a specified user.
2	/key	DELET E	This implementation of the DELETE operation deletes a specified Key&Secret.
3	/key	GET	This implementation of the GET operation re- turn a list of all Key&Secrets owned by the authenticated sender of the request.
4	/v1/user	POST	This implementation of the POST operation creates a new user.
5	/v1/user	PUT	This implementation of the PUT operation up- dates the specified user information.
6	/v2/user	GET	This implementation of the GET operation compatibles with third-party user systems.
7	/user/userid	DELET E	This implementation of the DELETE operation deletes the specified user information.
8	/users	GET	This implementation of the GET operation re- turn a list of all user information.
9	/share	POST	Create a specified data sharing policy.
10	/share	DELET E	Revoke a specified data sharing policy.
11	/share	PUT	Modify the specified data sharing policy.
12	/share/userid	GET	Get the list of data sharing policies for the specified user.

3. Experiments

In this section, we conduct experiments on real server to evaluate the feasibility of the MDSS framework. The MDSS is a framework of multi-user data sharing scheme bases on object storage in the cloud. This framework has no restrictions on the underlying object storage provider. You can use any object storage service as the underlying OSP of MDSS, and we use the open source service Minio as the OSP. In this experiment, we use the virtual machine as the test host for multi-user data sharing service. Each virtual machine rep- resents a personal host for the user. We use a server to provide users with virtual machine services. After have compared several widely-used virtualization technologies such as KVM [10], XEN [2], Hyper-V [13] and LXC [4][9], according to the actual situation, we finally chose to use KVM as a virtualization technology to provide services for users.

In the Alone-user mode or in the Multi-user mode, this framework separates data storage services from user personal hosts. The Alone-user mode focuses on improving the reliability and availability of data, while the Multi-user mode solves the problem of sharing data with permission on the alone-machine basis. The MDSS is already compatible with the key AWS S3 APIs, so users can apply the s3fs tool to mount buckets to the personal hosts and users can manipulate the data as if it was a local disk. When data owner shares his/her bucket with another user, the MDSS will record information about shared data, including shared permission, shared time limit and so on. The MDSS will verify the user’s operation request according to the recorded information, which can prevent data leakage and ensure data security and reliability.

This experiment was developed using Django, an open source python web application framework. The database design of the experiment is shown in Fig 3. The database is designed as three data tables: s3 buckets, s3 users and s3 shares. Data table s3 buckets records information about bucket, including creation time, creator information and so on. Data table s3 users records information about users, including access key, secret, creator ID and creation time etc. Data table s3 shares records permission information for data sharing between users, including shared bucket information, owners and sharers information, permission and deadline etc. To begin with, after receiving a user's operation request, the MDSS extracts the users Access from the request header. According to Access, the corresponding Key&Secret and authorization information is found in the database, and the signature is calculated. If the two signatures are the same, the user identity is legal. Furthermore, the module performs permission verification on the user's operation request. The platform executes the requested operation only when the operation request permission is less than or equal to the owned permission.

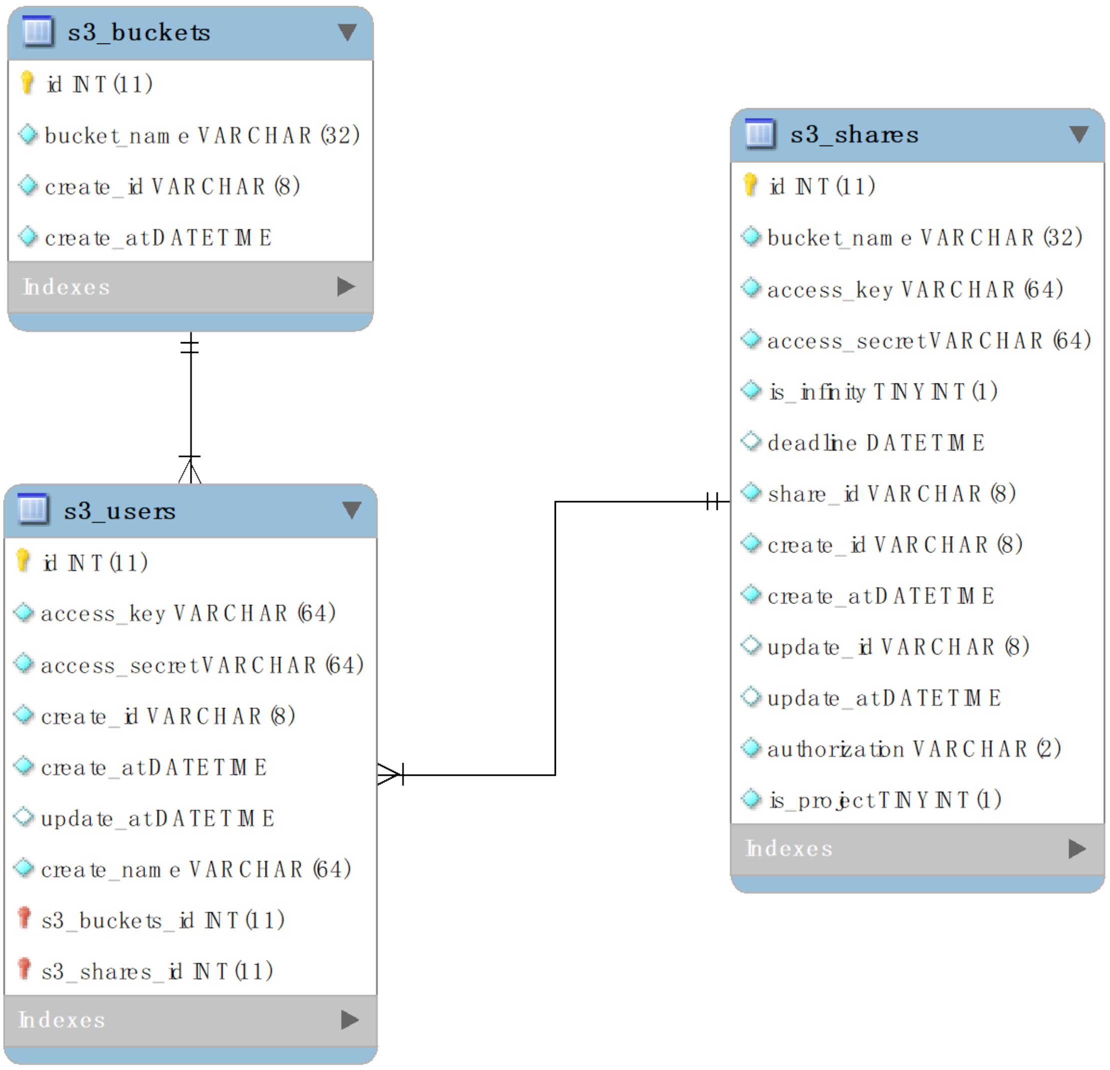


Fig. 3 The database design of the experiment

The API module of MDSS is divided into two parts, the compatible module and user operation module. The compatible module is developed on the basis of the AWS S3 REST API documentation. User operation module is developed according to the actual needs of users, with the focus on Key&Secret management and data sharing operation. User identity and operation authentication are implemented in combination with AWS Signature Version 4 and database records.

4. Results and Analysis

This experiment uses real server as management node and three virtual machines as test hosts for MDSS. The bucket in the MDSS can be mounted to the personal host by using the s3fs tool. S3fs automatically calculates AWS Version 4 signatures and sends requests to the server based on user actions, waiting for the server to respond. When the bucket in the MDSS is successfully mounted, the user can manipulate the data in the bucket as if it was a local disk. Postman tool can verify that the API conforms to the specification. Test 1 is a requested test for the GET Service API, adding the corresponding Key&Secret, AWS Region and Service Name information, the postman will automatically generate a standard request header according to the signature calculation method. The response information of a legitimate GET Service request is shown in Test 1. Note that the response of request must be the same as AWS S3 API documentation.

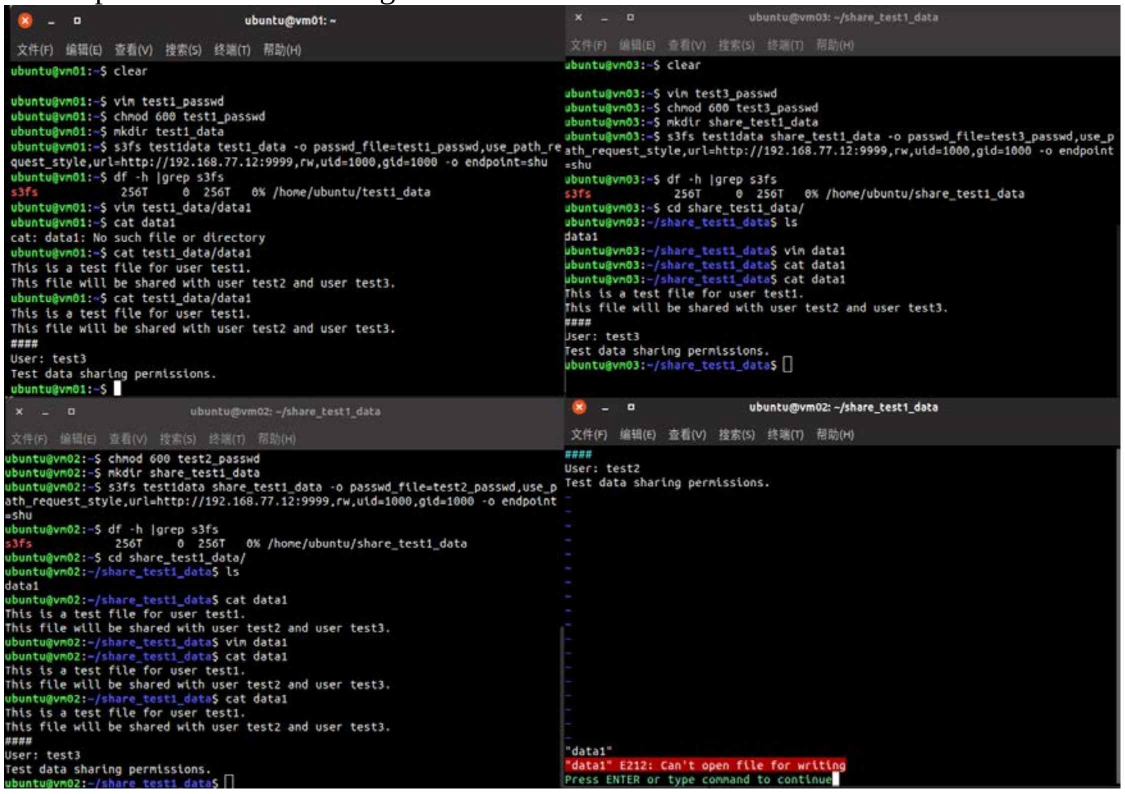
Test1	A legitimate GET Service request
Request:	GET / HTTP/1.1 Content-Type: application/x-www-form-urlencoded Host: 127.0.0.1:8000 X-Amz-Content-Sha256: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 X-Amz-Date: 20180822T012433Z Authorization: AWS4-HMAC-SHA256 Credential=ds3test/20180822/shu/s3/aws4request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date, Signature=2c212cf16e1403498fa30694973efd8415c63a7f7c45e1a3ec79a3075a41d182
Response:	<?xml version=”1.0” encoding=”UTF-8” ?> <ListAllMyBucketsResult> <Owner> <ID>17721803</ID> <DisplayName></DisplayName> </Owner> <Buckets> <Bucket> <Name>ds3test</Name> <CreationDate>2018-05-08T02:39:07.099896+00:00</CreationDate> </Bucket> </Buckets> </ListAllMyBucketsResult>

After implementing the MDSS, User can mount Bucket to personal host using the s3fs. When the mount command is executed, client can use the df command to check if the bucket has been successfully mounted. The successful mount will display that the available space is 256T. The client can manipulate the data in the Bucket as if it was a local disk. Each operation request executes the authentication module and the operation permission authentication module, and only the authenticated request platform will respond. Therefore, when private data is shared between users, the data user must have a reasonable level of authority to operate the bucket. Otherwise, the request will not be answered. The response information of an illegal GET Service request is shown in Test 2.

Test2	A illegal GET Service request
Request:	GET / HTTP/1.1 Content-Type: application/x-www-form-urlencoded Host: 127.0.0.1:8000 X-Amz-Content-Sha256: e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855 X-Amz-Date: 20180822T021751Z

	Authorization: AWS4-HMAC-SHA256 Credential=error/20180822/shu/s3/aws4 request, SignedHeaders=content-type;host;x-amz-content-sha256;x-amz-date, Signature=dc6d56a63f40b51657379d27d9626455cc39db0bd26bdd0b51d4a1f 593f65936
Response:	<?xml version=”1.0” encoding=”UTF-8” ?> <AuthenticationFailed> <Error> <Code>AccessDenied</Code> <Message>AccessDenied</Message> </Error> </AuthenticationFailed>

The three test hosts for the data sharing experiment were vm01, vm02 and vm03. Host vm01 is the owner of bucket ‘test1data’ and creates a data sharing policy. The shared data operation permission set by vm02 is GET, and vm03 is DELETE. The three test hosts mount bucket ‘test1data’ to the local using their respective Access&Secret messages. Because vm02 has only readable permissions, it is not possible to modify the shared data. Host vm03 has permission to modify data. The experimental results prove that the three hosts can efficiently perform real-time data synchronization. The experimental process is shown in Fig 4.


```
ubuntu@vm01: ~
ubuntu@vm01:~$ clear

ubuntu@vm01:~$ vim test1_passwd
ubuntu@vm01:~$ chmod 600 test1_passwd
ubuntu@vm01:~$ mkdir test1_data
ubuntu@vm01:~$ s3fs test1data test1_data -o passwd_file=test1_passwd,use_path_re
quest_style,url=http://192.168.77.12:9999,rw,uid=1000,gid=1000 -o endpoint=shu
ubuntu@vm01:~$ df -h |grep s3fs
s3fs            256T     0  256T   0% /home/ubuntu/test1_data
ubuntu@vm01:~$ vim test1_data/data1
ubuntu@vm01:~$ cat data1
cat: data1: No such file or directory
ubuntu@vm01:~$ cat test1_data/data1
This is a test file for user test1.
This file will be shared with user test2 and user test3.
ubuntu@vm01:~$ cat test1_data/data1
This is a test file for user test1.
This file will be shared with user test2 and user test3.
####
User: test3
Test data sharing permissions.
ubuntu@vm01:~$
```

```
ubuntu@vm03: ~/share_test1_data
ubuntu@vm03:~$ clear

ubuntu@vm03:~$ vim test3_passwd
ubuntu@vm03:~$ chmod 600 test3_passwd
ubuntu@vm03:~$ mkdir share_test1_data
ubuntu@vm03:~$ s3fs test1data share_test1_data -o passwd_file=test3_passwd,use_p
ath_request_style,url=http://192.168.77.12:9999,rw,uid=1000,gid=1000 -o endpoint
=shu
ubuntu@vm03:~$ df -h |grep s3fs
s3fs            256T     0  256T   0% /home/ubuntu/share_test1_data
ubuntu@vm03:~$ cd share_test1_data/
ubuntu@vm03:~/share_test1_data$ ls
data1
ubuntu@vm03:~/share_test1_data$ vim data1
ubuntu@vm03:~/share_test1_data$ cat data1
ubuntu@vm03:~/share_test1_data$ cat data1
This is a test file for user test1.
This file will be shared with user test2 and user test3.
####
User: test3
Test data sharing permissions.
ubuntu@vm03:~/share_test1_data$
```

```
ubuntu@vm02: ~/share_test1_data
ubuntu@vm02:~$ chmod 600 test2_passwd
ubuntu@vm02:~$ mkdir share_test1_data
ubuntu@vm02:~$ s3fs test1data share_test1_data -o passwd_file=test2_passwd,use_p
ath_request_style,url=http://192.168.77.12:9999,rw,uid=1000,gid=1000 -o endpoint
=shu
ubuntu@vm02:~$ df -h |grep s3fs
s3fs            256T     0  256T   0% /home/ubuntu/share_test1_data
ubuntu@vm02:~$ cd share_test1_data/
ubuntu@vm02:~/share_test1_data$ ls
data1
ubuntu@vm02:~/share_test1_data$ cat data1
This is a test file for user test1.
This file will be shared with user test2 and user test3.
ubuntu@vm02:~/share_test1_data$ vim data1
ubuntu@vm02:~/share_test1_data$ cat data1
This is a test file for user test1.
This file will be shared with user test2 and user test3.
ubuntu@vm02:~/share_test1_data$ cat data1
This is a test file for user test1.
This file will be shared with user test2 and user test3.
####
User: test3
Test data sharing permissions.
ubuntu@vm02:~/share_test1_data$
```

```
ubuntu@vm02: ~/share_test1_data
####
User: test2
Test data sharing permissions.
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
"data1"
"data1" E212: Can't open file for writing
Press ENTER or type command to continue
```

Fig. 4 Experimental process

5. Conclusion

In this work, we propose the MDSS method to address the security problem of sensitive personal data in data sharing. The MDSS enables secure data storage to introduce the AWS S3 signature calculation algorithm and operation the data of Bucket like a local disk. In general, the MDSS separates the user personal host from the data storage service, which can solve the problems of data recovery, data migration and limited storage space of the personal host. Data user can realize real-time synchronize data by mounting the same Bucket on multiple hosts. Data owner can set the permission information when sharing personal data, such as operation permission, time limit, resharing limit, host limit. The MDSS will validate the request and only respond to legitimate requests to ensure greater user data security. The MDSS is a framework for addressing data sharing security without restricting the OSP.

Acknowledgements

This work was supported by the National Key Research and Development Program of China under Grant No.2017YFE0117500.

References

[1] Arnold J (2014) Openstack swift: Using, administering, and developing for swift object storage. " O'Reilly Media, Inc."

[2] Barham P, Dragovic B, Fraser K, Hand S, Harris T, Ho A, Neugebauer R, Pratt I, Warfield A Xen and the art of virtualization. In: ACM SIGOPS operating systems review, 2003. vol 5. ACM, pp 164-177.

[3] Barton M, Reese W, Dickinson JA, Payne JB, Thier CB, Holt G (2015) Method for handling large object files in an object storage system. Google Patents.

[4] Bernstein D (2014) Containers and cloud: From lxc to docker to kubernetes. IEEE Cloud Computing (3):81-84.

[5] Cui H, Deng RH, Li Y (2018) Attribute-based cloud storage with secure provenance over encrypted data. Future Generation Computer Systems 79:461-472.

[6] GibsonGA, VanMeterR (2000) Networkattachedstoragearchitecture. Communications of the ACM 43 (11):37-45.

[7] Hashem IAT, Yaqoob I, Anuar NB, Mokhtar S, Gani A, Khan SU (2015) The rise of big data on cloud computing: Review and open research issues. Information systems 47:98-115.

[8] zrailevsky Y, Bell C (2018) Cloud Reliability. IEEE Cloud Computing 5 (3):39-44.

[9] Joy AM Performance comparison between linux containers and virtual machines. In: Computer Engineering and Applications (ICACEA), 2015 International Conference on Advances in, 2015. IEEE, pp 342-346.

[10] Kivity A, Kamay Y, Laor D, Lublin U, Liguori A kvm: the Linux virtual machine monitor. In: Proceedings of the Linux symposium, 2007. Dttawa, Dntorio, Canada, pp225-230.

[11] Kumar R, Gupta N, Charu S, Jain K, Jangir SK (2014) Open source solution for cloud computing platform using OpenStack. International Journal of Computer Science and Mobile Computing 3 (5):89-98.

[12] Leinenbach D, Santen T Verifying the Microsoft Hyper-V hypervisor with VCC. In: International Symposium on Formal Methods, 2009. Springer, pp 806-809.

[13] LiJ,ZhangY,ChenX,XiangY (2018) Secure attribute-based data sharing for resource limited users in cloud computing. Computers & Security 72:1-12.

[14] Li R, Shen C, He H, Gu X, Xu Z, Xu C-Z (2018) A lightweight secure data sharing scheme for mobile cloud computing. IEEE Transactions on Cloud Computing 6 (2):344-357.

[15] Mazumdar P, Agarwal S, Banerjee A (2016) Azure Architecture. In: Pro SQL Server on Microsoft Azure. Springer, pp 19-34.

[16] NadonJ (2017) Your Content Solution: An Introduction to AWS S3. In:Website Hosting and Migration with Amazon Web Services. Springer, pp 15-24.

[17] Ning J, Cao Z, Dong X, Liang K, Wei L, Choo K-KR (2018) CryptCloud+: secure and expressive data access control for cloud storage. IEEE Transactions on Services Computing.

[18] O'connor MA (2003) Method of enabling heterogeneous platforms to utilize a universal file system in a storage area network. Google Patents.

[19] Rawal BS, Vijayakumar V, Manogaran G, Varatharajan R, Chilamkurti N (2018) Secure disintegration protocol for privacy preserving cloud storage. Wireless Personal Communications:1-17.

[20] Reddin T, Kelleher LN, Coles A, Edwards A (2015) Persistent volume at an offset of a virtual block device of a storage server. Google Patents.

[21] Ren K, Wang C, Wang Q (2012) Security challenges for the public cloud. IEEE Internet Computing 16 (1):69-73.

[22] Shen J, Zhou T, Chen X, Li J, Susilo W (2018) Anonymous and traceable group data sharing in cloud computing. IEEE Transactions on Information Forensics and Security 13 (4):912-925.

[23] Shetty J, Anala M, Shobha G (2015) An approach to secure access to cloud storage service. International Journal of Research 2 (1):364-368.

[24] ThomasFC,WalkerPM,LipinskiGJ (2017)Direct attached storage system and method for implementing multiple simultaneous storage schemes. Google Patents.

[25] XueK, ChenW, LiW, HongJ, HongP (2018) Combining Data Owner-Sideand Cloud-Side Access Control for Encrypted Cloud Storage. IEEE Transactions on Information Forensics and Security 13 (8):2062-2074.