American-Eurasian Journal of Scientific Research 12 (2): 69-75, 2017
ISSN 1818-6785

DOI: 10.5829/idosi.aejsr.2017.69.75

Secure Blind Storage with Multiple User Access Provision

P. Golda Jeyasheeli, Nadar Jasmine Sunderraj and N. Umakanth

Department of Computer Science and Engineering, Mepco Schlenk Engineering College, Sivakasi, India

Abstract: In cloud computing, the client has to secure their private data from the server. To solve the concern, a storage scheme is used called as a Blind Server. Blind Server will act as a secured cloud storage service such that it does not know how the files are organized neither the size of each file nor the data in the file. The server will be free from computation overhead and will only act as a storage server. Two level security has been given to the data stored in the cloud by the client. In the first level, data has been encrypted and stored in the cloud. In the second level, the file is split into multiple fixed size block and stored in pseudo-random location in the hard disk so that the server will not be able to learn about which are the blocks that make up a file. This model of storage system is simulated on CloudSim. In the prior work, there is no information maintained about the blocks that contains the various splits of a single file and also there is no multiple user access provision. In the proposed scheme, the information about the blocks that hold the data of a file in the hard disk is stored in an array which is maintained by the client. By this mechanism, we have reduced the time taken for accessing a file by downloading and decrypting only the blocks that constitute the file along with multiple user access provision for the data stored in the cloud storage.

Key words: Blind Server · CloudSim · Computation overhead · A storage scheme · Pseudo-random location

INTRODUCTION

Now-a-days people have become habitual in outsourcing their data from their personal devices to the cloud servers because of the shortage of storage space in their personal devices like mobiles, personal computers, laptops and tablets. Therefore an additional responsibility is added on the client side to secure their sensitive data from the servers which might turn to be an in genuine one. To achieve this, a higher level security is given to the data in the cloud. The first level security implies encrypting the data and then storing in the cloud. The second level security is to store the data in arbitrary locations in the server. The Blind Storage scheme proposed by Naveed *et al*. [1] is found to be a secured storage scheme which provides security to the data stored in the server against honest but curious servers. But the time required to access a file on the client side is influenced by a sequence of download and decrypt operations where decryption is found to be a computationally intensive task. Therefore our main contribution is to reduce the time required to access a file by the client. This is achieved by reducing the number of download and decryption operations to be performed by the client. This model is simulated on the CloudSim framework.

CloudSim provides the essential entities to simulate a storage server which helped us in constructing the blind server model.

Related Work: Multi-Authority Attribute-Based Encryption (ABE) system is proposed in [1-2]. The different security challenges in a Public cloud is discussed in [3]. There are many ways for storing data in the cloud in a secured way which is dealt in [4-6]. An identity based hierarchical model for cloud computing and a corresponding encryption scheme is introduced in [7]. A high performance, distributed ORAM-based cloud data store is proposed in [8]. A dynamic searchable symmetric encryption scheme has been introduced in [9]. The integrity of the data stored in the server has to be verified by the client, for which a security mediator has been discussed in [10]. The process of uploading data to the

Corresponding Author: N. Umakanth, Department of Computer Science and Engineering, Mepco Schlenk Engineering College, Sivakasi, India.

server has motivated researchers to design a secured data storage on the server. Naveed *et al*. [1] has designed such a scheme where the file is split into multiple fixed size blocks and the file that has been split into multiple files occupy multiple blocks in the storage array. The similar Blind Storage scheme is also used in [11]. But there is no data collaboration service proposed for the blind storage scheme in the existing work. A data collaboration scheme for a group of users has been proposed in [12] which enables the file to be shared among a group of users. During the process of upload, a longer sequence of pseudo-random numbers are generated using a seed from which a subset is generated which consists of only free blocks and the size of the subset is equal to the size of the file in terms of number of blocks required to store the file. The file is finally stored in those blocks.

During the download operation, the pseudo-random sequence is again generated using the seed and a minimum of k blocks have to be downloaded and decrypted in every transaction. Then the decrypted blocks will be matched for file id, when the first block containing the file id is found the size of the file is retrieved from this block and the remaining blocks have to be downloaded and decrypted until all the blocks of the file has been found. This process involves an overhead of downloading and decrypting blocks that does not constitutes of the file to be accessed. Decryption is a computation intensive operation on the client and if avoiding the download and decryption of unwanted blocks is achieved then the time required to access a file on the client side can be significantly improved.

System Design: A hard disk consists of many blocks. The file to be stored on the cloud is split into multiple files of a fixed size such that the size depends on the size of the block. In general the block size varies from 256 bytes to 512 bytes. But the size of the block can also be fixed to a higher range by which the number of downloads will be minimized.

The file is split into multiple files of equal size (size of split depends on the size of the block in a hard disk) then these splitted files are stored in random block locations in the hard disk. The random locations are generated by setting a seed value.

Initially, for every file to be uploaded we will generate a File id and a key. These keys will be held by the client and will be useful for the download operation to be performed on the files stored in the server. The one who holds the key is seen as an authenticated client to download the data from the server. The key generation will take place only once for each file and has to be stored in the client for further access to the file.

The server does not have to perform any computation. It is free from computation overhead and will have the work of acting only as a storage server. All the computation is done on the client side. Hence the server will not be able to learn about which are the blocks that contains the various split of a single file as all the computation is done on the client side.

Blind Server: The model consists of a client and a server. The server will only be used for storage purposes. All the computation will be done by the client only. Fig. 1 represents the overall design of a Blind Server system.

The storage is constructed such that it consists of a number of blocks. Each block is of a fixed size. The file to be stored by the client on the server is split into many parts each split file will be equal to the block size. Each split file will be stored in the server in an unsystematic order so that the server will not have the knowledge about size of the file and about where the files are stored on the server. The contents of the file are encrypted therefore the server will not know about the file's contents.

The server is given the name blind because even though it stores the files it cannot have an access to the data in the files and it will neither know about which are the blocks that have to be merged to access a single file as the files splits are stored in an arbitrary order.

A storage array has to be constructed such that the array consists of B_n blocks. The size of the storage array should be fixed based on number of blocks needed for the files to be uploaded. In the prior work [1], the size of the array is set to be 4 times as many blocks as total data blocks to be stored.

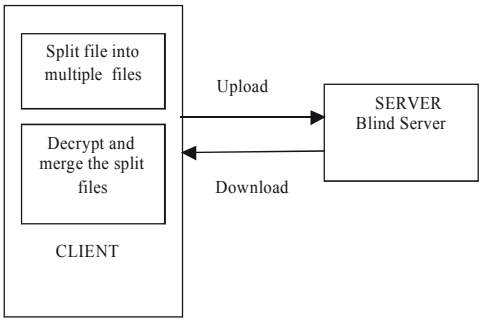

Fig. 1: Blind Server System

Client Side Computation: All the computation required to upload the files to the server and downloading the files from the server has to be performed on the client. The below given steps is a summary of the Blind Server algorithm. The algorithm will be discussed in detail in the next section. For each file the client wants to store in the server, the following steps have to be performed:

- Divide the file into equal parts according to the block size. For eg: If the block size is 512 bytes then the file has to be divided accordingly such that each file split is 512 bytes.
- Generate a file id for the file.
- Generate a key for the file.
- The file id and the key generated is to be stored on the client side.
- Set the seed. The seed value is the product of the file id and key.
- Using the seed generate an arbitrary sequence of bock numbers.
- Check which are the blocks that are empty among the sequence of blocks generated and take the first F_S blocks from the sequence generated. where, F_S is the size of the file in terms of blocks i.e the number of blocks required to store the complete file.
- Perform Encryption on the data to be stored in the cloud. The file to be stored on the cloud will contain encrypted data.
- If F_S blocks are found to be empty among the sequence generated then upload the files in those blocks.

Encryption and Decryption of data is done using the AES algorithm. Encryption is done before uploading the file on the cloud server and Decryption is done after retrieving the file from the server.

Let S be the storage array in the server that consists of B_n blocks of block size B_s each. Each block will store each split file. The split file will be containing encrypted content which will make the server unable to know the contents of the file.

The server will reinforce only two operations storing the data and allowing the client to retrieve the data. There will be no computation to be performed by the server and the server will be free from computation overhead.

Blind Server Algorithm: The client can store a set of files on this blind server. The server will be blind to see the content of the files and the size of the individual files. The Storage array S will consist of B_n blocks of b bits each. For each file to be stored on the server, do the following:

Storing the Files in the server:

- Split the file into equal size blocks.
- For every file among the set of files to be stored on the server generate file id and key which will be one time generation keys.
- Set the seed for generating arbitrary location of blocks in the storage array S.

$$\alpha = \text{File id} * \text{key} \quad (1)$$

- Depending on the number of blocks in the storage array S generate a lengthy sequence using the seed.
- α From the lengthy sequence take a subset of Q_0 block numbers.

$$Q_0 \subseteq B_n \quad (2)$$

The size of Q_0 is given by,

$$Max\ (\beta * F_s, \lambda) \quad (3)$$

where, β is the incremental parameter which is set to be greater than one.

λ, is the minimum number of blocks transmitted in every upload and download operation.

- From the arbitrary sequence generated by the seed α, Q_0 will take the n unique numbers and the size of n= Size of(Q_0).
- The set Q_0 should be chosen such that it satisfies two conditions:
- A minimum of F_S blocks in the storage array S pointed out by the set Q_0 should be free.
- Minimum one block in the storage array S pointed out by set Q_1 should be free $Q_1 \subseteq Q_0$.

$$\text{Size of Q1} = \lambda \quad (4)$$

The generation of set Q_1 will be useful for downloading the files from the server.

- If the above two conditions is not satisfied then adjust your parameter α so that the condition is satisfied.

- Choose a set Q_2 in such a way that the block numbers pointed out by Q_2 are all free and that the size of $Q_2 = F_S$.
- Encrypt the data in the files.
- Store the F_S file splits onto the F_S blocks in the storage array.

Data-Sharing Among a Group Of Users: The data stored in the blind storage can be shared among a group of users. This is achieved by one to many encryption paradigm. The root user encrypts data with multiple recipients public keys and stores it into the cloud server. So only those intended recipients can decrypt the data using their own secret key. The root user only takes public keys of the recipients as input to encrypt the data.

The private key is given to the intended recipients from the root user. The private key is generated by the root user and provided to all the intended recipients. The root user picks a master key for itself. The master key is a random seed picked by itself. A public key is considered as the e-mail id. The root user generates private key for the other users with whom it wants to share the file with, generally they are known as the intended recipients. The private key is generated for the intended recipients by the root user as follows:

$$P_{user1} = P_{ruser} + M_{ruser} H_{pbkuser1} \quad (5)$$

wqhere, P_{user1} is the private key generated for user1, P_{ruser} is the private key of the root user, M_{ruser} is the master key of the root user and $H_{pbkuser1}$ is the hashed value of the public key of user1. In the same way the root user generates private keys for all the n users with whom the root user wants to share the file with. The Ciphertext is set as follows:

$$C = [mH_{pbhuser1}, mH_{ruser}, \lambda \oplus H(m), F \oplus H(\lambda)] \quad (6)$$

C is the Ciphertext, m is a secret value set as $m = H(\lambda, F)$ where F is the data in the file. λ is picked in a random way where $\lambda \in \{0, 1\}^n$.

Downloading Files from the Server: The client who wants to access the file from the server should hold the file id and the key generated for each file. The client refers to the root user and the intended recipients with whom the root user wants to share the file stored in the cloud storage. Any user without the file id and key will not be able to access the file from the server and will be treated as an unauthorized one. Compute the seed using the file id and key.

$$\alpha = \text{File id} * \text{key} \quad (7)$$

Now evaluate whether the seed is a valid one (i.e evaluate whether such a document has been uploaded in the server). For each file stored in the server an array is maintained which consists of the seed value and the block locations in the hard disk in which that particular file was uploaded. When the user wants to access a file the initial step is to generate the seed. The root user is the one who uploads the file to the blind storage initially. Therefore at the time of private key generation to the n users the root user should also send the seed value to the n users so that the file can be downloaded from these n users later when they want an access to it. Then the seed is evaluated and found whether it is a valid one. If such a seed value is not maintained in the array, then the seed is said to be an invalid one i.e such a file using that seed was not uploaded in the server. This may happen when an unauthorized user tries to access the file or when an authorized user gives wrong file id or key generated for that file.

If the seed value is found to be a valid one then the array maintained for that file will hold the seed value along with the pseudo-random locations for that file. The block locations in the hard-disk in which various splits of that file is stored can be retrieved from the array and then the file can be downloaded from the server. The downloaded files will then be decrypted and merged to form the original file. To decrypt the data the root user has to first calculate the λ. Then compute $Cp \oplus H(\lambda)$, the resultant will yield the file data F, where $Cp = F \oplus H(\lambda)$ and is the input for the decryption process. The intended recipients of the file will perform the decryption as follows: Firstly, computes the $H_{pbkuser}$. Then computes λ followed by the computation of $Cp \oplus H(\lambda) = F$. After accessing the file the file contents have to be encrypted and then upload the file to the server.

RESULTS AND DISCUSSIONS

The model of the blind server was implemented using CloudSim. CloudSim is used as a foundation for implementing the Blind Server scheme. NetBeans 8.0.2 IDE is used.

File Split: The first file named 'Cohen' is of size 8,880Kb. The file has been split into 10 multiple files. Nine files each of 977Kb and one file of 91Kb. The single file needs ten blocks in the hard disk to store the 10 multiple splits.

The type of documents that has been used are text documents. The hard disk was created such that it consists of 51 blocks. The blocks have been numbered from 0 to 50. Each block is of 1Mb each. A set of five files were uploaded. The files taken were of size (10Mb, 3Mb, 7Mb, 2Mb, 3Mb) shown in Table 1. Each uploading and downloading of files required only one round of communication whereas in prior scheme [1] two round of communication would be needed for large files.

To make efficient use of the blocks in the hard disk, the value of β was set to (3, 4, 5). Each time the parameter varied there was a difference in the number of blocks filled in the array for the same set of files. Table 2 gives the details of the uploaded files their respective size and the block position occupied by each file in the hard disk.

To effectively use all the blocks in the hard disk we uploaded 5 files each of 10 Mb. Now our hard disk consists of 51 blocks therefore among the 51 blocks, 50 blocks should contain files. But we found that when the value of β is varied the blocks in the hard disk is not utilized effectively. This is shown in Table 3 where the row values 10B, 9B represents 10 Blocks and 9 Blocks respectively (B represents the blocks in the hard disk).

We are uploading a group of files, there is no problem in uploading the first four files of 10 Mb each but when the fifth file (10 Mb) is uploaded it occupies only 9 to 4 blocks in the hard disk even when empty blocks are available in the hard disk. This happens because β plays a major role in declaring the size of the subset Q_0 which consists of pseudo-random locations i.e the randomly chosen block locations from the hard disk. In order to make efficient utilization of all the 51 blocks in the hard disk, we tested different values for β. The values used for β were (5,4,3,2). Table 3 discusses about the different values taken for β. The parameters should be chosen such that all the blocks in the hard disk are utilized efficiently i.e when empty blocks are available then they must be used effectively and the files have to be uploaded successfully. When empty blocks are not available and we wish to upload additional group of files into the server then the size of the storage array in the server has to be increased. In the prior work [1] they have proposed to set the value of the storage array to be 4 times as many blocks as the total blocks to be stored. But this will lead to overheads in the storage.

Before increasing the size of the storage array we have to make use of the storage space currently available by effectively setting the parameter value β. This can reduce the storage overhead to small extent. We can summarize that the larger the value of β, the more effectively the available blocks in the hard disk are used.

Table 1: Uploaded Files

File Name	Size (no of Blocks)
Cohen	10
Positive	3
CEFd	7
PB	2
CAEE	3

Table 2: Blocks occupied by each file in the hard disk

File Name	Size (no of Blocks)	Blocks Occupied
Cohen	10	35,42,30,29,7,34,45, 21,9,41
Positive	3	14,25,13
CEFd	7	43,22,1,27,37,39,2
PB	2	28,16
CAEE	3	8,18,49

Table 3: Variation in β value

Uploaded Files	β= 5	β = 4	β = 3	β =2
File 1	10B	10B	10B	10B
File 2	10B	10B	10B	10B
File 3	10B	10B	10B	10B
File 4	10B	10B	10B	10B
File 5	10B	9B	5B	4B

Table 4: Variations in the number of pseudo-random locations generated

Files Up-Loaded	Pseudo locations [52]	Pseudo locations [53]	Pseudo locations [54]	Pseudo locations [62]
File 1	10B	10B	10B	10B
File 2	10B	10B	10B	10B
File 3	10B	10B	10B	10B
File 4	10B	10B	10B	10B
File 5	8B	8B	8B	8B

The subset Q_0 is taken from a longer sequence generated. The size of this longer sequence should also be set efficiently. So, that the subset can be retrieved from it effectively. The size of the longer sequence was also tested with different values as shown in Table 4 to validate the variations in the blocks occupied. In Table 4, the column Pseudo- locations [52] indicates that the size of the pseudo-random sequence generated using the seed was 52 (52 pseudo-random locations have been generated, from which the subset Q_0 will be taken).

The β value was set to be 5 because using β as 5 we got effective results. We generated pseudo-random locations by varying the size of the superset from 52 to 61 for which the results weren't effective but when the size of the superset was fixed at 62 we got good results. With β=5 and Pseudo-locations [62], we were able to upload all the five files of 10 Mb each leaving only one block as empty among the 51 blocks in the hard disk. Further if the client wants to store additional files in the server, then the size of the storage array S has to be scaled.

This will reduce the storage overhead as the size of the storage array is scaled only after effective utilization of current blocks in the storage array(hard-disk) whereas in the prior work it is said to have the size of S to be 4 time as many blocks as the number of total data blocks to be stored.

Different values for the sequence and β is tested to check the variations in the blocks occupied. When there is still empty blocks available for files to be uploaded there is no need to scale the size of the storage array which can reduce the storage overhead. Hence the proposed scheme is said to be effective in terms of the time taken to access the file on the client side and in terms of storage overhead.

Communication Costs: All the encryption, decryption key and the key generated for each file will be stored in the client. Only the file to be stored and it's block location has to be given to the server. There will be only one round of communication during upload and download whereas in the prior scheme there will be two rounds of communication during the download operation when the size of the file is large. In the first round, λ blocks will be downloaded and if size of the file in terms of blocks is greater than λ then the remaining blocks have to be downloaded. These blocks are indexed by the pseudo-random set Q_0 from which the remaining blocks have to be downloaded.

In our scheme there will be only one round of communication during download as the information about which are the blocks that constitute a single file is stored in a data structure in the client. The information about which are the blocks to be downloaded are then sent to the server and those blocks can be downloaded from the server. The process will be repeated for downloading a set of files from the server. This will therefore reduce the time taken to access a file on the client.

Computation Overhead: As there is no computation to be done on the server, the server is free from computation overhead. The server will only act as a cloud storage service. All the computation will be done by the client. The expensive operation considered on the client is the encryption and decryption. But this overhead is also reduced by reducing the number of decryptions to be performed on the client during download while comparing to the previous work.

CONCLUSION

A secured cloud data storage was simulated on CloudSim. The Client will perform the encryption and the encrypted data is stored in pseudo-random locations in the server. The data stored in the cloud can be accessed by multiple users with the one-to-many encryption paradigm. The server is free from computation. This is done at the cost of minimum information leakage to the server thereby providing a secured cloud data storage. The future work would be to make the scheme secure against actively corrupt server.

REFERENCES

1. Naveed, M., M. Prabhakaran and C.A. Gunter, 2014. Dynamic searchable encryption via blind storage, in the Proceedings of the IEEE Symp. Secur. Privacy, pp: 639-654.
2. Lewko, A. and B. Waters, 2011. Decentralizing attribute-based encryption, in the Proceedings of the, EUROCRYPT. Berlin, Germany: Springer-Verlag, pp: 568-588.
3. Ren, Cong Wang and Qian Wang, 2012. Security Challenges for the Public Cloud Kui, IEEE Computer Society, pp: 69-73.
4. Roopa, Manjunath, 2013. Secure Way of Storing Data in Cloud Using Third Party Auditor, IOSR Journal of Computer Engineering (IOSR-JCE) e-ISSN: 2278-0661, p- ISSN: 2278-8727, 12: 69-74.
5. Anuradha, R. and Y. Vijayalatha, 2013. A Distributed Storage Integrity Auditing for Secure Cloud Storage Services, International Journal of Advanced Research in Computer Science and Software Engineering, 3(8).
6. Doshi Manasi and Swapnaja Hiray, 2013. Secure and Data Dynamics Storage Services on Cloud, International Journal of Advanced Research in Computer Science and Software Engineering, 3(11).
7. Li, H., Y. Dai, L. Tian and H. Yang, 2009. Identity based authentication for cloud computing, in Cloud Computing. Berlin, Germany: Springer-Verlag, pp: 157-166.
8. Stefanov, E. and E. Shi, 2013. Oblivistore: High performance oblivious cloud storage, in IEEE Security & Privacy, pp: 253-267.
9. Kamara S., C. Papamanthou and T. Roeder, 2012. Dynamic searchable symmetric encryption, in Proc. ACM, pp: 965-976.

10. Chaudhari Sanket, Akanksha Singh and Sheetal Asopa, 2015. Security Mediator Using Blind Signatures and Key Rotation Algorithm, International Journal of Advanced Research in Computer and Communication Engineering, 4(3).
11. Li Hongwei, Dongxiao Liu, Yuanshun Dai, Tom H. Luan and Xuemin Shen, 2015. Enabling Efficient Multi-Keyword Ranked Search Over Encrypted Mobile Cloud Data Through Blind Storage, IEEE Transactions on Emerging Topics in Computing, 3(1).
12. Xin Dong, Jiadi Yu, Yanmin Zhu, Yingying Chen, Yuan Luo and Minglu Li, 2015. SECO: Secure and scalable data collaboration services in cloud computing, Elsevier Computers and Security, pp: 91-105.