International Journal on Applications in Information and Communication Engineering
Volume 5 : Issue 1 : June 2019 , pp 55 –61 www.aetsjournal.com ISSN (Online) : 2394 - 6237

SECURE BLIND STORAGE WITH MULTIPLE USER ACCESS PROVISION

Bharathi S, Aravind A, Deepika E, Ganesh P, Shravan D
Department of Information Technology
Rathinam Technical Campus, Coimbatore, Tamilnadu, India

Abstract— Clients who use cloud computing have the responsibility of keeping their personal information secure from the server. In order to solve the problem, a storage strategy that is known as a Blind Server is put into action. Blind Server will operate as a safe cloud storage service, during which it will conceal any information regarding the file sizes, file types, or file organisation. The server won't have any computational overhead because it will only serve as a storage server for the organisation. The customer has added an additional layer of protection to the data that is stored in the cloud. The first level of protection for the data includes encrypting it and storing it in the cloud. In order to prevent the server from being able to identify which blocks belong to which files, the file is broken up into many blocks of a consistent size and then placed in a second level of the hard drive that is pseudo-randomly organised. This idea of a storage system is replicated on CloudSim. Previous work did not keep any information regarding the blocks that contained the various splits of a single file, and it did not permit multiple users to access the file at the same time. The proposed system contains a description of the information about the hard drive blocks that are responsible for holding a file's data, and this information is saved in an array that the client is responsible for maintaining. We were able to shorten the amount of time necessary to access a file by utilising this strategy, which involved merely downloading and decrypting the individual blocks that comprise the file. Additionally, this method enabled a large number of users to access the information that was kept in the cloud storage.

Index Terms— Computational time, cloud computing services Local Binary Pattern (LBP),.

I. INTRODUCTION

People now routinely outsource the storage of their data from personal devices, such as smartphones, laptops, personal computers, and tablets, to remote servers located in the cloud. This is necessary because people's personal devices have limited storage space. Because of this, it is now the responsibility of the customer to safeguard the confidentiality of any sensitive data stored on servers that may not be authentic. In order to accomplish this, a higher level of protection is applied to the data that is stored in the cloud. Encryption of data and storing it in the cloud are two necessities for the initial level of protection. As a secondary layer of protection, the data may be stored in any of the server's available locations. It has been determined that the Blind Storage method that was introduced by Naveed et al. [2] is a secure storage scheme that safeguards the data that is kept in the server from being accessed by servers that are trustworthy but inquisitive. [Citation needed] On the client side, the amount of time needed to access a file is influenced by the order in which download and decryption operations are performed. Decryption is found to be a task that requires a significant amount of computational resources. As a consequence of this, our primary contribution is to shorten the amount of time required for a customer to access a particular file. This is accomplished by reducing the number of tasks the client is required to perform in terms of downloading and decrypting data. The simulation of this model is carried out by utilising the CloudSim framework. With the help of CloudSim, which provides all of the components that are required to simulate a storage server, we were able to construct the model of the blind server.

II. CONNECTED WORK

[5] suggests using a system known as Multi-Authority Attribute-Based Encryption, or ABE for short.

[8] discusses the many different security issues that can arise when using a public cloud. There are a variety of safe ways to store data in the cloud, some of which are discussed in [10]–[13]. An identity-based hierarchical paradigm for cloud computing is presented in [5,] along with the associated encryption method. A suggestion for a decentralised, high-performance cloud data storage that makes use of ORAM can be found in [7]. A dynamically searchable symmetric encryption system was described in reference [6], which you can find here. It is the responsibility of the client to verify the veracity of the data that has been stored on the server; [9] has presented the idea of using a security mediator for this objective. The procedure of uploading data to the server provided the academics with the impetus to design a safe and reliable data storage system for use on the server.

Naveed and his colleagues [2] came up with this method, which involves slicing the file up into a number of different blocks of a predetermined size, with each of these blocks taking up a distinct location in the storage array. [1] makes use of a method that is conceptually analogous to Blind Storage. On the other hand, the earlier research does not make any recommendations regarding a data collaboration service for the blind storage strategy. A plan for collaborating on data has been presented in [3, and as a result, the file can be made available for sharing among a number of different users. During the upload process, a longer sequence of seemingly random numbers is produced with the help of a seed. A subset is created out of this longer sequence that only includes free blocks and is the same size as the file in terms of the number of blocks required to store the file. This subset is called a free block subset. These blocks are where the file is finally stored after it has been opened.

Every transaction needs to download and decrypt at least k blocks, and the pseudo-random sequence is generated once more during the downloading process by using the seed. Once the first block that contains the file id has been located, the size of the file can be determined using the information contained in this block. After this, the remaining blocks of the file must be downloaded and decrypted one at a time until all of the blocks of the file have been located. After that, the decrypted blocks are going to be compared to the file id. As part of this process, it is necessary to download and decrypt any blocks that do not belong to the file that is currently being viewed. Because decryption is a computationally demanding operation on the client, the amount of time it takes to access a file on the client side can be significantly reduced if unnecessary blocks are not downloaded and decrypted.

III. DESIGN OF SYSTEMS

Numerous blocks constitute the structure of a hard drive. The file that will be stored in the cloud is divided into several files, each of which has a fixed size whose size is determined by the size of the block. The size of the file that will be stored in the cloud is determined by the size of the block. The size of a block can range anywhere from 256 to 512 bytes, on average. On the other hand, the block size can be increased to a higher range in order to cut down on the number of downloads.

The file is then split into several files of the same size; the size of the split is determined by the size of a block on the hard disc. These files are then saved in arbitrary block locations on the hard disc. The random locations can be generated by specifying a seed value.

In the beginning, we are going to generate a File id and a key for every single file that will be handed in. These keys, which are required in order to carry out the download operation on the files that are stored on the server, will be in the possession of the client. The person who has the key is considered an authenticated client and can therefore download the data from the server. Before the client can access the file again, the key must first be saved in the client's memory, as the key will only be generated once for each individual file.

The server does not have to carry out any calculations because it is not necessary. Its sole purpose will be to operate as a storage server, and it will not be responsible for any overhead tasks associated with computing. All of the computation is finished on the client side. Due to the fact that all computation takes place on the client's end, the server will be unable to determine which blocks within a single file contain the various splits that have been made.

The BLIND SERVER

The model consists of both a server and a client component. On the server, there will be no operations other than storage. The entirety of the computation will be carried out solely by the client. Figure 1 illustrates the overall configuration of a system that uses a Blind Server.

The storage space is organised in such a way that it is constructed out of a number of separate blocks. The dimensions of each block are predetermined. The client's copy of the file that will be stored on the server has been cut up into multiple parts, each of which will have a size that is equivalent to the block size. Each split file will be stored on the server in an arbitrary order, which will prevent the server from being able to determine either the size of the file or the location on the server where the files are located. The server won't be able to view the encrypted contents of the file because those contents are protected by encryption.

The server is referred to as "blind" due to the fact that, despite the fact that it stores the files, it is unable to view the contents of the files and is unable to determine which blocks need to be combined in order to access a single file due to the fact that the divided files are stored in an arbitrary order.

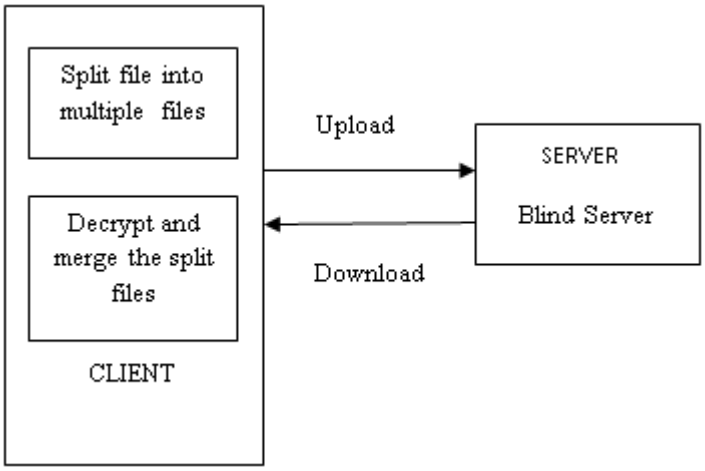

Fig 1: Blind Server System

It is essential to construct a storage array in such a way that it incorporates Bn blocks. The size of the storage array ought to be set in stone, and it should be determined by the number of blocks essential for the uploading of the files. In the earlier work [2], the size of the array was determined to be four times as many blocks as there are total data blocks that are to be stored. This setting is still in place.

B. Customer-Side Computing

When a client wants to upload files to the server or retrieve files from the server, it is the client's responsibility to carry out all of the necessary computation. The steps that follow provide an explanation of the algorithm that underlies the Blind Server. In the following section, additional information regarding the algorithm will be discussed in depth. The client is required to carry out the steps outlined in the following paragraphs for each and every file that they wish to store on the server:

1. Determine the size of the blocks to be used as a guide for dividing the file into equal parts. If the size of each block is 512 bytes, for example, the file needs to be split up in such a way that each of the resulting pieces is also 512 bytes.

2. Create a unique identifier for the file.

3. Generate a key that can be used to access the file.

4. The client's side is the only place where the generated key and file id can be saved.

5. Start by planting the seed Multiplying the file id by the key yields the seed value, which is generated in this manner.

6. Make a random sequence of block integers by using the seed as a starting point.

7. Determine which blocks in the sequence of blocks that were generated are empty, and then choose the first FS blocks that are found in that sequence.

where FS represents the total number of blocks required to store the file, also known as the size of the file expressed in terms of blocks.

8. Encrypt any information that is going to be stored in the cloud. The file that is stored in the cloud storage service will contain data that will later be encrypted.

9. After the sequence has been formed, upload the files into any empty FS blocks that may still be available.

The Advanced Encryption Standard (AES) is used to encrypt and decrypt data. Encryption takes place prior to a file being uploaded to a cloud server, and decryption takes place after the file has been downloaded from the server.

Let's say that the storage array on the server is denoted by the letter S. It is composed of Bn blocks, each of which has a size of Bs, and each divided file is kept in its own individual block. The server will not be able to decipher the contents of the split file because it contains encrypted data that prevents it from doing so.

The server will only improve performance for two operations: storing the data and making it available to clients so they can get it. There will be no need for any computation to be carried out on the server, and there will also be no overhead associated with computing.

Algorithm C. BLIND SERVER

The client has the option of storing their files on the blind server at all times. The server won't be able to see the sizes of the individual files or the contents of the files themselves. Every single block that makes up the storage array S will consist of b bits. Carry out the tasks listed below with regard to each file that will be stored on the server:

Files being kept on the server

1. Divide the file into blocks of equal size.
2. Create a file id and key, which will be one-time generation keys, for each file among the group of files to be saved on the server.
3. Establish the seed for the storage array S's random block generation.
4. = File id + key (1)
5. Using the seed, create a long sequence that varies in length depending on how many blocks are in the storage array S.
6. Select the Q0 block numbers that make up a portion of the lengthy sequence.

a. $$Q_0 \subseteq B_n$$

(2)

b. The size of Q_0 is given by,

c. $$Max(\beta * F_S, \lambda)$$

(3)

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

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

7. 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).
8. The set Q_0 should be chosen such that it satisfies two conditions:
9. A minimum of F_S blocks in the storage array S pointed out by the set Q_0 should be free.
10. Minimum one block in the storage array S pointed out by set Q_1 should be free $Q_1 \subseteq Q_0$.
    i. Size of Q1= λ (4)
    ii. The generation of set Q_1 will be useful for downloading the files from the server.
11. If the above two conditions is not satisfied then adjust your parameter α so that the condition is satisfied.
12. 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 .
13. Encrypt the data in the files.
14. Store the F_S file splits onto the F_S blocks in the storage array.

The exchange of data between different user groups The data that is kept in the blind storage can be shared by one person or multiple people. This is something that can be done thanks to the one-to-many encryption paradigm. Data is encrypted on the cloud server by the root user, and the public keys of multiple recipients are used to decrypt the data later. Because of this, only the people who are supposed to receive the data can decrypt it by using their own private key. When encrypting the data, the root user only makes use of the public keys belonging to the recipients. The private key is disseminated to the appropriate recipients by the user with root privileges. The private key is generated by the root user, and it is then distributed to each of the intended recipients individually. The root user decides which key should be the master key. A seed is selected at random and is used to calculate the master key. People often think of their email address as a public key. The root user will produce a private key for each of the other users with whom it plans to share the file. These other users are referred to as the intended receivers. The private key for the intended recipients is generated by the root user in the following manner:

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

Where Puser1 is the private key that was generated for user1, Pruser is the private key that belongs to the root user, Mruser is the master key that belongs to the root user, and Hpbkuser1 is the value that was hashed from user1's public key. In the same way, the root user must generate private keys for each of the n other users before sharing the file with any of them. The Ciphertext is arranged in this manner:

$$C = [mH_{pbkuser1}, mH_{ruser}, \lambda \oplus H(m), F \oplus H(\lambda)] \quad (6)$$

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 $\lambda \in \{0,1\}^n$.

Getting Files Off the Server

The client who wishes to access the file from the server is responsible for storing the file id as well as the key that was generated for each file. The client refers to the root user as well as any other users with whom they wish to share a file that they have stored in the cloud storage service. Anyone who accesses the file from the server without the file id and key will be regarded as an unauthorised user and will not be permitted to access the file. Calculating the seed requires using the file id as well as the key.

= File id + key (7)

Find out now whether the seed can be relied upon (i.e evaluate whether such a document has been uploaded in the server). An array containing the seed value as well as the block addresses on the hard disc location where a specific file was uploaded is maintained for every file that is stored on the server. When a user wants to access a file, the first thing they need to do is create the seed for the file. The "root" user is the one who initiates the process of uploading the file to the "blind" storage. In order for the file to be downloaded from these n users in the future when they desire access to it, the root user should give the seed value to the n users at the same time as sending the private key to the n users. This will allow the file to be downloaded from these n users. After that, the seed is examined to determine whether or not it is viable. If that kind of seed value had not been kept in the array, the file in question would not have been able to be uploaded to the server, which would have rendered the file invalid. This could take place if a user who is not authorised to access the file attempts to do so, or if a user who is authorised to access the file enters an incorrect file id or key that was produced for that file.

If it is determined that the seed value is a valid one, it will be saved in the array for that file along with the file's seemingly random locations if it is determined that it is a valid one. After using the array to retrieve the block addresses on the hard drive where different sections of the file are stored, the file can then be downloaded from the server. This occurs after the file has been used to retrieve the different sections of the file. After that, the encrypted files that were downloaded will be joined together, and the result will be the original file. Prior to being able to decode the data, the root user is required to first compute. After that, the data for the file denoted by the letter F will be produced by computing

where Cp= and serving as the input for the decryption procedure. The following decryption will be carried out by the recipients to whom the file is addressed: begins by computing the value of. The computation of comes after that, followed by the computation of. Once access has been gained to the file, the contents of the file need to be encrypted before they can be uploaded to the server.

IV. RESULT AND CONVERSATION

CloudSim was utilised in order to put the blind server model into action. CloudSim is used as the foundation for the implementation of the Blind Server. It is built with the NetBeans IDE version 8.0.2.

split files

The first document bearing the name "Cohen" takes up 8,880 KB of storage space. The file was split up into ten different files altogether. There are nine files, each of which is 977 KB in size, and one file that is 91 KB. In order for the hard disc to be able to store all ten multiple splits in a single file, it requires ten blocks.

Files Uploaded in Table I

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

The type of documents that have been utilised thus far are known as text documents. The assembled hard drive is comprised of a total of 51 blocks. The numbers 0 through 50 are written on the blocks. The size of each block is 1 megabyte. A total of five files were uploaded at the same time. The size of the files that were obtained corresponded to what was shown in Table I. (10Mb, 3Mb, 7Mb, 2Mb, and 3Mb). In contrast to the previous method [2], which necessitated two rounds of communication for large files, the current method [3] necessitated only one round of communication for the uploading and downloading of a file respectively.

In order to make the most efficient use of the space available on the hard disc, the value of was changed to (3, 4, 5). The number of blocks that were completed in the array for the same collection of files changed depending on the argument that was passed in. The information regarding the uploaded files can be found in Table II. This information includes the file sizes as well as the locations of each file's individual blocks on the hard drive. In order to make effective use of the space available on the hard disc, we uploaded five files totaling 10 megabytes each. Given that our hard drive has 51 blocks, it is reasonable to expect that 50 of those blocks will contain data files. However, we found out that when the value of is changed, the blocks on the hard disc are not properly used in the way that they should be. This is demonstrated in Table III, where the row values 10B and 9B denote the number of blocks as 10, and 9, respectively (B represents the blocks in the hard disk).

FILE NAME	SIZE(NO OF BLOCKS)
Cohen	10
Positive	3
CEFd	7
PB	2
CAEE	3

Table II Blocks occupied by each file in the hard disk

When we upload a collection of files, the first four files, each of which is 10 megabytes, do so without any problems; however, when we upload the fifth file, which is also 10 megabytes, it only takes up 9 to 4 blocks on the hard drive, even if there are vacant blocks available. This occurs regardless of whether there are vacant blocks available. This takes place regardless of whether there are empty blocks that can be utilised. This occurs as a result of having a significant impact on the size declaration for the subset Q0, which is composed of pseudo-random locations, or randomly selected block locations from the hard disc. This occurs as a result of having a significant impact on the size declaration for the subset Q0. This is a result of having a significant impact on the size declaration, which causes this to take place. We tried out a wide range of different configurations in order to maximise the use of the storage space on the hard disc. The factors that were considered in order to (5,4,3,2). A discussion of the many different values that were utilised can be found in Table III. It is essential that the parameters be configured in such a way that every free block on the hard disc is used to its utmost capacity, so it is essential that this be done properly.

This indicates that whenever there are free blocks, those blocks have to be used to their full potential in order for the files to successfully upload. This must be done whenever there are free blocks. When there are no empty blocks available on the server, the size of the storage array on the server needs to be increased so that it can accommodate the uploading of larger groups of data. This is because there are no empty blocks available on the server. They suggested that the value of the storage array be set to be four times as many blocks as the total number of blocks that had to be stored in their earlier work [2]. This was their recommendation. On the other hand, as a direct result of this, the costs associated with storage will go up.

The parameter value needs to be correctly set before we can increase the capacity of the storage array. This is necessary so that we can make the best possible use of the storage space that is already available to us. It's possible that this will only have a very slight positive impact on the storage overhead. In conclusion, as the value of increases, the available blocks on the hard disc are utilised in a manner that is more effective than before. This occurs because of the way that the hard disc is organised.

The subset Q0 is constructed out of the longer sequence that was produced, which can be found here. In addition to this, it is of the utmost importance to calculate the duration of this longer sequence in an accurate manner. so that the subset can be extracted from it in a manner that is both efficient and effective. As can be seen in Table IV, a variety of approaches to determining an appropriate value for the length of the extended sequence were tested. This was done so that the differences in the occupied blocks could be verified and analysed. The length of the supposedly random sequence that was produced by using the seed was 52, as can be seen in the Pseudo-locations column of table IV. The seed was used to generate the sequence. (Fifty-two supposedly random locations have been generated, and it is from these that we will select the subset Q0.)

Table III - 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

Because doing so produced useful results, the value was set to 5, which was the default. When the size of the superset was held constant at 62, we were able to generate pseudo-random locations successfully by varying the size of the superset from 52 to 61. This led to the best results. We were only able to leave one block empty out of the total of 51 blocks on the hard drive after uploading all five files that were 10 megabytes each by using =5 and pseudo-locations [62]. In addition, if the customer wishes to save more information on the server, they will need to increase the size of the storage array denoted by the letter S. In contrast to earlier research, which suggested that the size of S should be four times as large as the number of total data blocks that need to be stored, this will result in a reduction of storage overhead because the size of the storage array will only be scaled after the effective utilisation of the blocks that are currently contained in the storage array (hard-disk).

We look at the sequence and all of its different values to see how the differences in the blocks that are occupied change over time. When there are still empty blocks available for files to be uploaded, it is not necessary to increase the size of the storage array because this will only serve to increase the storage overhead. As a consequence of this, the strategy that was suggested is evaluated as successful in terms of the client side access time to the file and the storage overhead.

A. Costs of communication

All of the encryption, decryption, and key information that was generated for each file will be saved on the client. The only thing that needs to be given to the server is the file that is going to be stored and the location of its block. As opposed to the previous scheme, which required two rounds of communication whenever a large file was downloaded, the new one will only require one communication round during uploading and downloading of files. Blocks will be downloaded during the first round, and if the file's block size is greater than a certain threshold, then the remaining blocks will need to be downloaded as well. It will be necessary to download the remaining blocks from the pseudo-random set Q0, which serves as an index for these blocks.

Our method for downloading files only needs one round of communication because the client is responsible for storing in a data structure the information regarding which blocks comprise a file. After that, the server is made aware of the blocks that require downloading, and it is then possible to download those blocks from the server itself.

The procedure is carried out once more each time a different collection of files needs to be downloaded from the server. Because of this, getting access to a file on the client will take significantly less time.

Table IV - 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

B. The costs associated with computation

There is no need for the server to carry out any computation because there is none that needs to be carried out there. Cloud storage will be the only function that the server will perform. The customer is responsible for carrying out all calculations. The most expensive processes for the customer to take into account are the ones involving encryption and decryption of data. In contrast to previous research, however, this burden is reduced further by reducing the number of client-side decryptions that are required during the download process.

V. Conclusion

A safe cloud-based data storage environment was simulated on CloudSim. The client will be responsible for performing the encryption, while the server will be responsible for storing the encrypted data in supposedly random locations. Numerous users are able to access the data that is kept in the cloud storage facility thanks to the one-to-many encryption paradigm. Computing tasks are not performed on the server. Because of this, the cloud data storage is secured, and the server experiences the minimum amount of information leakage that is technically possible. The primary focus of any future development work will be on finding ways to make the scheme secure against servers that are deliberately corrupt.

REFERENCES

[1]. "Enabling Efficient Multi-Keyword Ranked Search Over Encrypted Mobile Cloud Data Through Blind Storage," IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, March 2015, Volume 3 number 1. Tom H. Luan, Yuanshun Dai, Hongwei Li, and Xuemin Shen.

[2]. "Dynamic searchable encryption using blind storage," in Proc. IEEE Symp. Secur. Privacy, May 2014, pp. 639–654.

[3]. SECO: Secure and scalable data collaboration services in cloud computing, Elsevier computers & security 2015, p. 91-105, by Xin Dong, Jiadi Yu, Yanmin Zhu, Yingying Chen, Yuan Luo, and Minglu Li

[4]. "Identity based authentication for cloud computing," in Cloud Computing, by H. Li, Y. Dai, L. Tian, and H. Yang 2009, p. 157–166, Berlin, Germany: Springer-Verlag.

[5]. "Decentralizing attribute-based encryption," by A. Lewko and B. Waters, in Proc. EUROCRYPT, Berlin, Germany: Springer-Verlag, 2011, pp. 568–588.

[6]. "Dynamic searchable symmetric encryption," in Proc. ACM, 2012, pp. 965–976. S. Kamara, C. Papamanthou, and T. Roeder

[7]. "Oblivistore: High performance oblivious cloud storage," in IEEE Security & Privacy, 2013, pp. 253-267, by E. Stefanov and E. Shi

[8]. Security Challenges for the Public Cloud Kui, Ren, Cong Wang, and Qian Wang, IEEE Computer Society, 2012, pp. 69–73.

[9]. ecurity Mediator Using Blind Signatures and Key Rotation Algorithm, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 4, Issue 3, March 2015. Sanket Chaudhari, Akanksha Singh, and Sheetal Asopa.

[10]. Manjunath Roopa, "Secure Way of Storing Data in Cloud Using Third Party Auditor," Volume 12 of the IOSR Journal of Computer Engineering (IOSR-JCE), July-August 2013, pages 69-74.

[11]. International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 8, August 2013, Anuradha.R, and Dr. Y. Vijayalatha, "A Distributed Storage Integrity Auditing for Secure Cloud Storage Services."

[12]. Secure and Data Dynamics Storage Services on Cloud, Manasi Doshi and Swapnaja Hiray, International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 11, November 2013.

[13]. "Storing Shared Data in the Cloud through Security-Mediator," Xidian University, Xi'an, China. Boyang Wang, Sherman S. M. Chow, Ming Li, and Hui Li.