2012 International Conference on Information and Network Technology (ICINT 2012)
IPCSIT vol. 37 (2012) © (2012) IACSIT Press, Singapore

A Scheme for Checking Data Correctness in the Cloud

Malathi.M [1 +] and Murugesan.T [2]

[1] Indra Ganesan College of Engineering,Trichirapalli,TamilNadu
[2] Pavai College of Engineering,Namakal,TamilNadu

Abstract. Reducing costs, accelerating processes and simplifying management are all vital to the success of an effective IT infrastructure. Companies are increasingly turning to provide more flexible IT environments to help them realise these goals. One such solution is Cloud Computing. Cloud Computing enables tasks to be assigned to a combination of software and services over a network. For example storage of large data in cloud reduces costs and maintenance. But the customer is unaware of the storage location. Here risk involved is modification of data or tampering of data. Since the customer does not have control over data the cloud provider should assure the customer that data is not modified. In this paper we propose a data correctness scheme in which a Third Party can audit the data stored in the cloud and assure the customer that the data is safe**.** This scheme ensures that the storage at the client side is minimal which will be beneficial for thin clients.

Keywords: availability, outsourcing, updation, deletion, modification.

1. Introduction

Small Scale Enterprises(SMEs) and Medium Scale Enterprises move their data or store their data in cloud storage centers to minimise the costs and maintenance. Data outsourcing brings with it many advantages. But associated with it is the risks involved. For example the end user or customer may not be knowing where the data stored. He loses his control over his data. So the necessary thing is to ensure the data from modification or alteration during the period of storage. So he appoints a Third Party Auditor to check the availability of data and its correctness. TPA verifies the data stored in the cloud and communicates this with the consumer(client). Whenever the client needs the data the cloud returns the data with full guarentee of delivery, availability and correctness. As TPA verifies for its correctness and availability he considers the data is safe.

In this paper we propose a scheme for Integrity and availability of data. Such a scheme is useful for peer-to-peer storage systems, network file systems, long term archives, web-service object stores, and database systems.Such verification systems prevent the cloud storage archives(storage) from misrepresenting or modifying the data stored by using frequent checks on the storage archives. And cloud user should frequently does this periodically to ensure that his data is available and correct at anytime.

2. Related Work

Data Integrity Proofs in Cloud Storage by Sravan kumar and Asthosh Saxena provides a scheme for static storage of data.[1]. Juels et al. [2] described a formal "proof of retrievability"(POR) model for ensuring the remote data integrity. Their scheme combines spot-checking and error-correcting code to ensure both possession and retrievability of files on archive service systems. Shacham et al. [3] built on this model and

+ Corresponding author. Tel.: + 04312714678.
E-mail address: malathimurugesan@hotmail.com.

constructed a random linear function based on homomorphic authenticator which enables unlimited number of queries and requires less communication overhead. Bowers et al. [4] proposed an improved framework for POR protocols that generalizes both Juels and Shacham's work. Later in their subsequent work, Bowers et al. [5] extended POR model to distributed systems. However, all these schemes are focusing on static data.

3. Proposed Scheme

We propose a data correctness scheme which involves the encryption of the few bits of data per data block thus reducing the computational overhead on the clients. This is based on the fact that high probability of security can be achieved by encrypting fewer bits instead of encrypting the whole data. The client storage overhead is also minimized as it does not store any data with it and it reduces bandwidth requirements. Hence our scheme suits well for small memory devices and low power devices. In our data integrity protocol the TPA needs to store only a single cryptographic key irrespective of the size of the data file F and two functions which generate a random sequence. The TPA does not store any data with it. The TPA before storing the file at the archive, preprocesses the file and appends some meta data to the file and stores at the archive. At the time of verification the TPA uses this meta data to verify the integrity of the data. It is important to note that our proof of data integrity protocol just checks the integrity of data. But the data can be stored, that is duplicated at redundant data centers to prevent the data loss from natural calamities.

If the data has to be modified which involves updation, insertion and deletion of data at the client side ,it requires an additional encryption of fewer data bits. So this scheme supports dynamic behaviour of data.

4. Correctness Proof by Selecting Random Bits in Data Blocks

The TPA first selects fewer bits of the entire file and preprocesses the data. This fewer bits constitute metadata. This meta data is encrypted and appended to the file and sent to the cloud. Then whenever the client needs to verify the data correctness and availability it challenges the cloud through TPA and the data it got is correct, then integrity is ensured. This scheme can be extended for data updation, deletion and insertion at the client side. This involves modification of fewer bits at the client side.

There are two phases. One is Setup phase and the other is verification phase. Setup phase include generation of metadata and its encryption. Verification phase includes issuing a challenge to the Cloud server and getting a response and checking its validity.

4.1. Setup phase

Let the verifier V wishes to the store the file F with the archive. Let this file F consist of n file blocks. We initially preprocess the file and create metadata to be appended to the file. Let each of the n data blocks have m bits in them. A typical data file F which the client wishes to store in the cloud is shown in fig 1. The initial setup phase can be described in the following steps.

The data file is named as F.

Number of blocks n in each file is n.

Each block comprises of m bits.

k number of bits out of m bits of n blocks are selected for the construction of Meta data.

1	2	3	4	...	...	...	n

Fig. 1: Data Blocks

1	2	3	4	...	...	...	n
m bits	m bits	m bits	m bits	m bits	m bits	m bits	m bits

Fig. 2: k bits of each block < m bits of each block

1) Generation of meta-data: Let g be a function defined as follows.

$$g(i, j) \longrightarrow \{1..m\}, \quad i\epsilon\{1,..n\}, \quad j\,\epsilon\{1,..k\}$$

Where k is the number of bits per data block which we wish to read as meta data. The function g generates for each data block a set of k bit positions within the m bits that are in the data block. Hence g(i, j) gives the jth bit in the ith data block. The value of k is the choice of the TPA and is a secret known him only. For each data block we get a set of k bits and in total for all the n blocks we get n * k bits. Let m_i represent the k bits of meta data for the ith block.

2) Encrypting the meta data: Each of the meta data from the data blocks m_i is encrypted by using a suitable algorithm to give a new modified meta data M_i. Let h be a function which generates a k bit integer $\boldsymbol{\alpha_i}$ for each i. This function is a secret and is known only to the TPA.

$$h : i \longrightarrow \boldsymbol{\alpha_i}, \boldsymbol{\alpha_i} \epsilon \{ 0..2^n \}$$

For the meta data (m_i) of each data block the number $\boldsymbol{\alpha_i}$ is added to get a new k bit number M_i.

$$M_i = m_i + \boldsymbol{\alpha_i}$$

In this way we get a set of n new meta data bit blocks. The encryption method can be improved to provide still stronger protection for data.

3) Appending of meta data: All the meta data bit blocks are generated using the above procedure .Now they are concatenated together. This concatenated meta data should be appended to the file F before storing it at the cloud server. The file F when it is appended with meta data becomes F' This file is archieved with the cloud. Fig. 3 shows the encrypted file F' after appending the meta data to the data file F.

Data File	MetaData

Fig. 3: The encrypted file F' after appending the meta data to the data file F

4.2. Verification phase

Let the TPA wants to verify the integrity of the file F. It throws a challenge to the archive and asks it to respond. The challenge and the response are compared and if the result is TRUE the TPA accepts the integrity proof. Else if the result of comparison is FALSE it rejects the integrity proof. Suppose the verifier wishes to check the integrity of ith block the TPA challenges the cloud storage server by specifying the block number i and a bit number j generated by using the function g which only the TPA knows. The TPA also specifies the position at which the meta data corresponding to the block i is appended. This meta data will be a k-bit number. Hence the cloud storage server is required to send the bits for verification by the client. The meta data sent by the cloud is decrypted by using the number $\boldsymbol{\alpha_i}$ .The corresponding bit in this decrypted meta data is compared with the bit that is sent by the cloud. Any mismatch between the two would mean a loss of the integrity of the clients data at the cloud storage.

5. Support for Data Dynamics

This scheme supports modification of data , that is data updation, data insertion and data deletion.

5.1. Data insertion

Data insertion is the most frequently used operations in cloud data storage. This refers to inserting new blocks after some specified positions in the data file F.Suppose insert few more blocks then number of blocks becomes $n'$from n. Since number of blocks are more now the meta data is updated. So no of bits m and no of metabits k becomes more. New function g is generated using the equation,

$$g(i_i, j_i) \longrightarrow \{1..m_i\},$$

Where $i_i \in \{1..n'\}$, $j_i \in \{1..k_i\}$,

$$h : i_i \longrightarrow \alpha'_i \ , \ \alpha'_i \in \{ 0.. 2^{n'} \}$$

For the meta data of each data block the number α'_i is added to get a new k bit number $\mathbf{M'_i}$

$$\mathbf{M'_i} = m'\mathbf{i} + \alpha'_i$$

Client constructs an update request message and send it to the server. Upon receiving the request the Server runs execution. Execution is nothing but updation. If the client wants to know whether updation has been done. It can issue a challenge through TPA .The TPA issues a challenge and receives the response similar to the procedure stated in verification phase. And the client comes to know whether the data is updated.

5.2. Data deletion

Data deletion is just the opposite operation of data insertion. For single block deletion it refers to deleting the specified block and moving all the latter blocks forward. Similarly for single file operation all the files are moved forward if one file is to be deleted. Suppose the server receives the update request for deleting block ni, the server deletes the block, correspondingly the m bits and k number of bits are deleted. And new meta data is generated. The procedure is same as stated in the data insertion operation.

5.3. Data modification

A basic data modification operation refers to the replacement of specified blocks with new ones. A client replaces the block n_i with n_i' For the new blocks and new m bits, the k bits are generated. The procedure for generating the k bits are same as above and generating the function g,h are similar to the data insertion operation. If the client wants to know whether updation has been done. It can issue a challenge through TPA .The TPA issues a hallenge and receives the response similar to the procedure stated in verification phase. And the client comes to know whether the data is updated.

6. Conclusion and Future Works

In this paper we have proposed a scheme to facilitate the client in getting a proof of integrity of the data which he wishes to store in the cloud storage servers with minimum costs and efforts. This scheme reduces the computational and storage overhead of the client as well as the computational overhead of the cloud storage server. This scheme also minimized the size of the proof of data integrity so as to reduce the network bandwidth consumption. At the client we store only two functions, the bit generator function g, and the function h which is used for encrypting the data. Hence the storage at the client is very much minimal. Hence this scheme is useful for devices which have less memory. The operation of encryption of data generally consumes a large computational power. In our scheme the encrypting process is very much limited to only a fraction of the whole data thereby saving on the computational time of the client. Many of the schemes proposed earlier require the archive to perform tasks that generally consumes a large computational power. We are trying to improve the scheme for auditing multiple files from multiple clients simultaneously.

7. References

[1] Ashutosh Saxena,Sravan Kumar "Data Integrity Proofs in Cloud Storage" 978-1-4244-8953-4/11 ,2011 IEEE

[2] A. Juels and J. Burton S. Kaliski, "PORs: Proofs of Retrievability for Large Files," *Proc. of CCS '07*, pp. 584–597,2007

[3] H. Shacham and B. Waters, "Compact Proofs of Retrievability," *Proc. of Asiacrypt '08*, Dec. 2008

[4] K. D. Bowers, A. Juels, and A. Oprea, "Proofs of Retrievability: Theory and Implementation," Cryptology ePrint Archieve, Report/175,2008, http://eprint.iacr.org/.

[5] K. D. Bowers, A. Juels, and A. Oprea, "HAIL: A High-Availability and Integrity Layer for Cloud Storage," Cryptology ePrint Archive, Report 2008/489, 2008, http://eprint.iacr.org/.