INTERNATIONAL JOURNAL FOR DEVELOPMENT IN COMPUTER SCIENCE & TECHNOLOGY ISSN-2320-7884 (Online)
www.ijdcst.com ISSN-2321-0257 (Print)

Data Integrity Proofs in Cloud Storage

Y.V.BHASKAR REDDY M.C.A., M.Tech.(PhD).#[1] , R.vijaya,M.Tech#[2] P.Ramadevi M.Tech#[3]
#[1]Associate professor , #[2] Assistant Professor #[3] Assistant Professor

Abstract—Cloud computing has been envisioned as the de-facto solution to the rising storage costs of IT Enterprises. With the high costs of data storage devices as well as the rapid rate at which data is being generated it proves costly for enterprises or individual users to frequently update their hardware. Apart from reduction in storage costs data outsourcing to the cloud also helps in reducing the maintenance. Cloud storage moves the user's data to large data centers, which are remotely located, on which user does not have any control. However, this unique feature of the cloud poses many new security challenges which need to be clearly understood and resolved.

One of the important concerns that need to be addressed is to assure the customer of the integrity i.e. correctness of his data in the cloud. As the data is physically not accessible to the user the cloud should provide a way for the user to check if the integrity of his data is maintained or is compromised. In this paper we provide a scheme which gives a proof of data integrity in the cloud which the customer can employ to check the correctness of his data in the cloud. This proof can be agreed upon by both the cloud and the customer and can be incorporated in the Service level agreement (SLA). This scheme ensures that the storage at the client side is minimal which will be beneficial for thin clients.

I. INTRODUCTION

Data outsourcing to cloud storage servers is raising trend among many firms and users owing to its economic advantages. This essentially means that the owner (client) of the data moves its data to a third party cloud storage server which is supposed to - presumably for a fee - faithfully store the data with it and provide it back to the owner whenever required.

As data generation is far outpacing data storage it proves costly for small firms to frequently update their hardware whenever additional data is created. Also maintaining the storages can be a difficult task. Storage outsourcing of data to a cloud storage helps such firms by reducing the costs of storage, maintenance and personnel. It can also assure a reliable storage of important data by keeping multiple copies of the data thereby reducing the chance of losing data by hardware failures.Storing of user data in the cloud despite its advantages has many interesting security concerns which need to be extensively investigated for making it a reliable solution to the problem of avoiding local storage of data. Many problems like data authentication and integrity (i.e., how to efficiently and

securely ensure that the cloud storage server returns correct and complete results in response to its clients' queries [1]), outsourcing encrypted data and associated difficult problems dealing with querying over encrypted domain [2] were discussed in research literature.

In this paper we deal with the problem of implementing a protocol for obtaining a proof of data possession in the cloud sometimes referred to as Proof of retrievability (POR).This problem tries to obtain and verify a proof that the data that is stored by a user at a remote data storage in the cloud (called cloud storage archives or simply archives) is not modified by the archive and thereby the integrity of the data is assured. Such kinds of proofs are very much helpful in 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 from misrepresenting or modifying the data stored at it without the consent of the data owner by using frequent checks on the storage archives. Such checks must allow the data owner to efficiently, frequently, quickly and securely verify that the cloud archive is not cheating the owner. Cheating, in this context, means that the storage archive might delete some of the data or may modify some of the data. It must be noted that the storage server might not be malicious; instead, it might be simply unreliable and lose or inadvertently corrupt the hosted data. But the data integrity schemes that are to be developed need to be equally applicable for malicious as well as unreliable cloud storage servers. Any such proofs of data possession schemes do not, by itself, protect the data from corruption by the archive. It just allows detection of tampering or deletion of a remotely located file at an unreliable cloud storage server. To ensure file robustness other kind of techniques like data redundancy across multiple systems can be maintained.

While developing proofs for data possession at untrusted cloud storage servers we are often limited by the resources at the cloud server as well as at the client. Given that the data sizes are large and are stored at remote servers, accessing the entire file can be expensive in I/O costs to the storage server. Also transmitting the file across the network to the client can consume heavy bandwidths. Since growth in storage capacity has far outpaced the growth in data access as well as network bandwidth, accessing and transmitting the entire archive even occasionally greatly limits the scalability of the

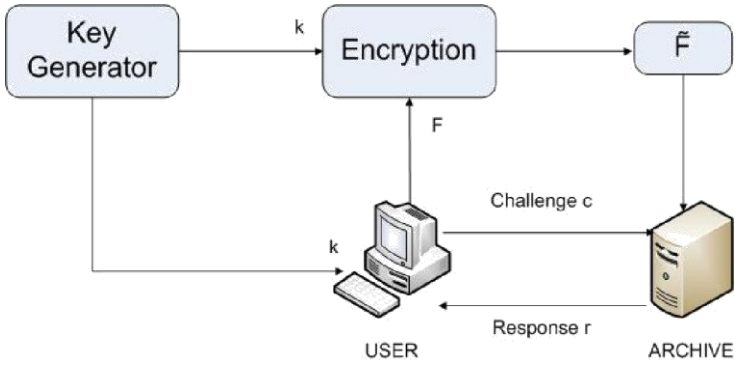

Fig. 1. **Schematic view of a proof of retrievability based on inserting random sentinels in the data file** F **[3]**

network resources. Furthermore, the I/O to establish the data proof interferes with the on-demand bandwidth of the server used for normal storage and retrieving purpose. The problem is further complicated by the fact that the owner of the data may be a small device, like a PDA (personal digital assist) or a mobile phone, which have limited CPU power, battery power and communication bandwidth. Hence a data integrity proof that has to be developed needs to take the above limitations into consideration. The scheme should be able to produce a proof without the need for the server to access the entire file or the client retrieving the entire file from the server. Also the scheme should minimize the local computation at the client as well as the bandwidth consumed at the client.

II. RELATED WORK

The simplest Proof of retrivability (POR) scheme can be made using a keyed hash function $h_k(F)$. In this scheme the verifier, before archiving the data file F in the cloud storage, pre-computes the cryptographic hash of F using $h_k(F)$ and stores this hash as well as the secret key K. To check if the integrity of the file F is lost the verifier releases the secret key K to the cloud archive and asks it to compute and return the value of $h_k(F)$. By storing multiple hash values for different keys the verifier can check for the integrity of the file F for multiple times, each one being an independent proof.

Though this scheme is very simple and easily implementable the main drawback of this scheme are the high resource costs it requires for the implementation. At the verifier side this involves storing as many keys as the number of checks it want to perform as well as the hash value of the data file F with each hash key. Also computing hash value for even a moderately large data files can be computationally burdensome for some clients(PDAs, mobile phones, etc). As the archive side, each invocation of the protocol requires the archive to process the entire file F . This can be computationally burdensome for the archive even for a lightweight operation like hashing. Furthermore, it requires that each proof requires the prover to read the entire file F - a significant overhead for an archive whose intended load is only an occasional read per file, were every file to be tested frequently[3].

Ari Juels and Burton S. Kaliski Jr proposed a scheme called Proof of retrievability for large files using ”sentinels”[3]. In this scheme, unlike in the key-hash approach scheme, only a single key can be used irrespective of the size of the file or the number of files whose retrievability it wants to verify. Also the archive needs to access only a small portion of the file F unlike in the key-has scheme which required the archive to process the entire file F for each protocol verification. This small portion of the file F is in fact independent of the length of F . The schematic view of this approach is shown in Figure 1.

In this scheme special blocks (called sentinels) are hidden among other blocks in the data file F . In the setup phase, the verifier randomly embeds these sentinels among the data blocks. During the verification phase, to check the integrity of the data file F , the verifier challenges the prover (cloud archive) by specifying the positions of a collection of sentinels and asking the prover to return the associated sentinel values. If the prover has modified or deleted a substantial portion of F , then with high probability it will also have suppressed a number of sentinels. It is therefore unlikely to respond correctly to the verifier.To make the sentinels indistinguishable from the data blocks, the whole modified file is encrypted and stored at the archive. The use of encryption here renders the sentinels indistinguishable from other file blocks. This scheme is best suited for storing encrypted files.

As this scheme involves the encryption of the file F using a secret key it becomes computationally cumbersome especially when the data to be encrypted is large. Hence, this scheme

proves disadvantages to small users with limited computational power (PDAs, mobile phones etc.). There will also be a storage overhead at the server, partly due to the newly inserted sentinels and partly due to the error correcting codes that are inserted. Also the client needs to store all the sentinels with it, which may be a storage overhead to thin clients (PDAs, low power devices etc.).

III. OUR CONTRIBUTION

We present a scheme which does not involve the encryption of the whole data. We encrypt only few bits of data per data block thus reducing the computational overhead on the clients.

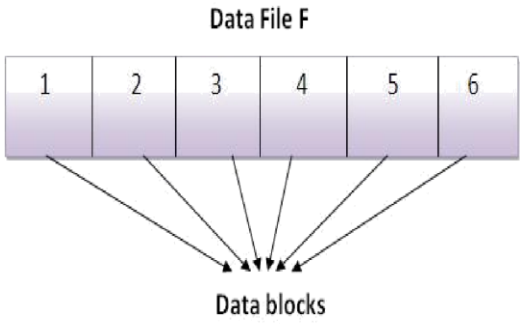

Fig. 2. **A data file** F **with 6 data blocks**

The client storage overhead is also minimized as it does not store any data with it. Hence our scheme suits well for thin clients.

In our data integrity protocol the verifier 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 verifier does not store any data with it. The verifier 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 verifier 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 i.e. if the data has been illegally modified or deleted. It does not prevent the archive from modifying the data. In order to prevent such modifications or deletions other schemes like redundant storing etc, can be implemented which is not a scope of discussion in this paper.

IV. A DATA INTEGRITY PROOF IN CLOUD BASED ON SELECTING RANDOM BITS IN DATA BLOCKS

The client before storing its data file F at the client should process it and create suitable meta data which is used in the later stage of verification the data integrity at the cloud storage. When checking for data integrity the client queries the cloud storage for suitable replies based on which it concludes the integrity of its data stored in the client.

A. 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 Figure 2. The initial setup phase can be described in the following steps

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

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

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 j^{th} bit in the i^{th} data block. The value of k is in the choice of the verifier and is a secret known only to him. Therefore 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 i^{th} block. Figure 3 shows a data block of the file F with random bits selected using the function g.

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.

Without loss of generality we show this process by using a simple XOR operation. Let h be a function which generates a k bit integer α_i for each i. This function is a secret and is known only to the verifier V .

$$h : i \rightarrow \alpha_i,\ \alpha_i \in \{0..2^n\} \quad (2)$$

For the meta data (m_i) of each data block the number α_i is added to get a new k bit number M_i.

$$M_i = m_i + \alpha_i \quad (3)$$

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

3) Appending of meta data: All the meta data bit blocks that are generated using the above procedure are to be con-catenated together. This concatenated meta data should be appended to the file F before storing it at the cloud server. The file F along with the appended meta data F^e is archived with the cloud. Figure 4 shows the encrypted file F^e after appending the meta data to the data file F .

B. Verification phase

Let the verifier V want 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 the verifier accepts or rejects the integrity proof.

Suppose the verifier wishes to check the integrity of n^{th} block. The verifier challenges the cloud storage server by

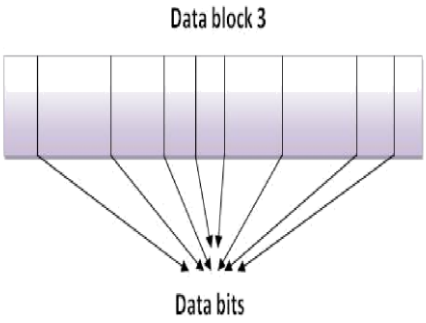

Fig. 3. A data block of the file F with random bits selected in it

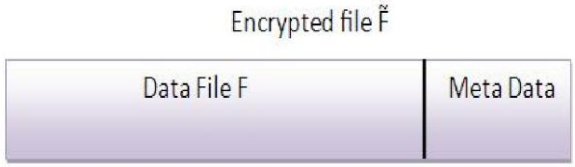

Fig. 4. **The encrypted file** F **which will be stored in the cloud**.

specifying the block number i and a bit number j generated by using the function g which only the verifier knows. The verifier also specifies the position at which the meta data corresponding the block i is appended. This meta data will be a k-bit number. Hence the cloud storage server is required to send k+1 bits for verification by the client.

The meta data sent by the cloud is decrypted by using the number α_i and 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.

V. CONCLUSION AND FUTURE WORKS

In this paper we have worked to facilitate the client in getting a proof of integrity of the data which he wishes to store in the cloud storage servers with bare minimum costs and efforts. Our scheme was developed to reduce the computational and storage overhead of the client as well as to minimize the computational overhead of the cloud storage server. We also minimized the size of the proof of data integrity so as to reduce the network bandwidth consumption.

At the client we only store 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 compared to all other schemes [4] that were developed. Hence this scheme proves advantageous to thin clients like PDAs and mobile phones.

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 need a lot of computational power to generate the proof of data integrity[3]. But in our scheme the archive just need to fetch and send few bits of data to the client.

The network bandwidth is also minimized as the size of the proof is comparatively very less(k+1 bits for one proof).

It should be noted that our scheme applies only to static storage of data. It cannot handle to case when the data need to be dynamically changed. Hence developing on this will be a future challenge. Also the number of queries that can be asked by the client is fixed apriori. But this number is quite large and can be sufficient if the period of data storage is short. It will be a challenge to increase the number of queries using this scheme.

REFERENCES

[1] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," Trans. Storage, vol. 2, no. 2, pp. 107–138, 2006.

[2] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," in SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy. Washington, DC, USA: IEEE Computer Society, 2000, p. 44.

[3] A. Juels and B. S. Kaliski, Jr., "Pors: proofs of retrievability for large files," in CCS '07: Proceedings of the 14th ACM conference on Computer and communications security. New York, NY, USA: ACM, 2007, pp. 584–597.