Proofs For Data Integrity In Cloud Storage

Kanmani.P[1], Anusha.S[2]

K.S.Rangasamy College Of Technology
Tiruchengode
[1]pkanmaniit@gmail.com

K.S.Rangasamy College Of Technology
Tiruchengode
[2]sekharanusha967@gmail.com

Abstract— **Cloud computing provides the genuine solution to the rising storage costs of IT Enterprises. Enterprises and individual users find it difficult to frequently update their hardware due to increasing costs of data storage and rapid increase in data rate. Data outsourcing to the cloud also helps in reducing the maintenance apart from reduction of storage costs. Cloud storage moves the user's data to large data centres that 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 is data integrity 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 a scheme is provided to obtain 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).**

Keywords— **Cloud computing, integrity, security.**

I. Introduction

Cloud computing provides unlimited infrastructure to store and execute customer data and program. As customers they do not need to own the infrastructure, they are merely accessing or renting; they can forego capital expenditure and consume resources as a service, paying instead for what they use. Data outsourcing to cloud storage servers is an increasing trend among many firms and users due to its economic advantages. This essentially means that the owner (client) of the data moves the data to a third party cloud storage server which is supposed to store the data apparently for a fee 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. [1] Data outsourcing to the cloud helps such firms by reducing the costs of storage, maintenance and personnel. Security concerns arise since both customer data and program are residing in provider premises. Security is always a major concern in Open System Architectures. When user uses the cloud, user probably won't know exactly where your data is hosted. Data should be stored and processed only in specific jurisdictions as define by user.[5] Provider should also make a contractual commitment to obey local privacy requirements on behalf of their customers.[2]

In this paper a protocol for obtaining a proof of data possession in the cloud is implemented referred 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 is not modified by the archive and thereby the integrity of the data is assured.. Such verification systems prevent the cloud storage archives from misrepresenting or altering the data stored at it without the permission of the data owner by conducting frequent checks on the storage archives. Such checks must allow the data owner to verify that the cloud archive is not cheating the owner. Cheating, in this context, means that some malicious storage server might delete some of the data. It must be noted that the storage server must be reliable. 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 does not, protect the data from corruption by the archive. It just allows detection of altering or deletion of a remotely located file at an unreliable cloud storage server. Other techniques such as data redundancy across multiple systems can be maintained to ensure file robustness. We are often limited by the resources at the cloud server as well as at the client while developing proofs for data possession at untrusted cloud storage servers. Accessing the entire file in the storage server can be expensive due to I/O costs and transmitting the file across the network to the client can consume heavy bandwidths. The problem is further complicated by the fact that the owner of the data may be a small device, like a PDA or a mobile phone, which have limited CPU power, battery power and communication bandwidth. Hence a data integrity proof has to be developed needs to taking above limitations into consideration. A proof must be developed without the need for the user to access the entire file stored at the server. The proof should not consume heavy bandwidth. Data integrity refers to maintaining stability of the data stored in distributed platforms. Strict enforcement of data integrity rules causes the error rates to be lower, resulting in time saved troubleshooting and tracing erroneous data and the errors it causes algorithms. Three types of integrity constraints are an inherent part of the relational data model: entity integrity, referential integrity and domain integrity.

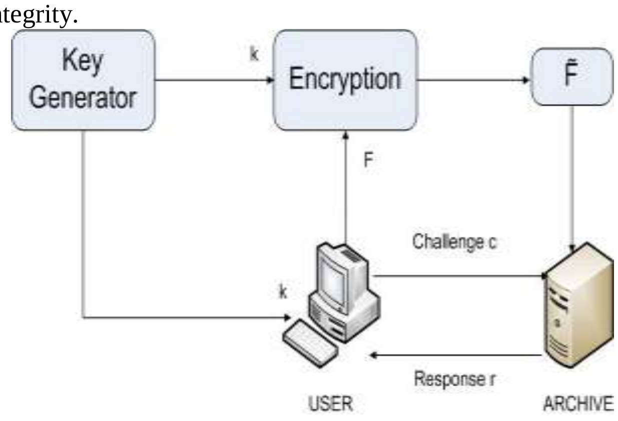

Fig. 1. Schematic views of a proof of retrievability based on inserting random sentinels in the data file F .

II. RELATED WORK

The simplest Proof of retrievability (POR) scheme is developed by using the keyed hash function h_k (F). In this scheme the client, pre-computes the cryptographic hash of F using h_k (F) and stores this hash as well as the secret key K before accessing the data file in the cloud server. The verifier then releases the secret key K to the cloud archive and asks it to compute and return the value of h_k (F).If both the values are same then the file integrity is not lost. The verifier can check for the integrity of the file F for multiple times, by storing multiple hash values for different keys each one being an independent proof. Though this scheme is very simple and easily implementable the main drawback of this scheme is that it requires high resource costs for implementation. At the server side, each invocation of the protocol requires the archive to process the entire file F which 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 [3].

Ari Juels and Burton S. Kaliski Jr proposed a scheme called Proof of retrievability for large files using "sentinels"[3]. This scheme is different from the key-hash approach. This scheme uses only a single key irrespective of the size of the file or the number of files whose retrievability it needs to verify. Also the archive needs to access only a small portion of the file F unlike in the key-hash 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 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 storage overhead to thin clients (PDAs, low power devices etc.)

III. PROPOSED WORK

Previously mentioned integrity schemes is likely to detect when the data stored in the archive is static and also the above mentioned protocols did not provide good results when number of queries has been increased. Taking above limitations into considerations a scheme has been proposed which supports both public auditability and dynamic data. The client before storing its data file F at the client should process it and create suitable metadata 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. 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. This proof can be agreed upon by both the cloud and the customer and can be incorporated in the Service level agreement (SLA).

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. The file is pre-processed and metadata is generated which is appended to the file. Let each of the n data blocks have m bits in them. The initial setup phase can be described in the following steps:

B. Generation of Metadata

Let g be a function defined as follows

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

where k is the number of bits per data block which we wish to read as metadata. 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 metadata for the i^{th} block. The process as in figure 2 is initiated with the generation of a public key parameter P_k by the cloud client. Then the client generates a signature for individual file blocks. The signature is a form of metadata which is a combination of public key and file blocks and are called as codes. Finally the generated metadata is transmitted to the cloud storage.[4]

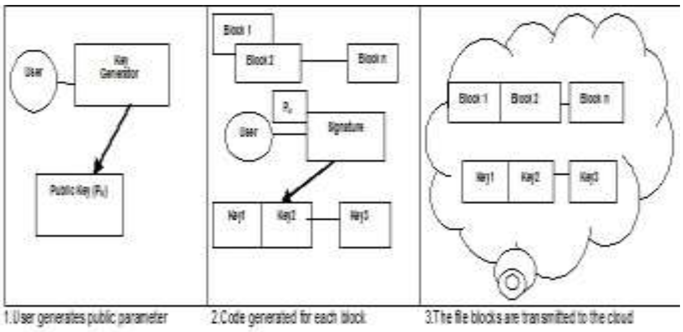

Fig. 2 Metadata Generation

C. Encrypting the Metadata

Each of the metadata from the data blocks mi is encrypted by using a suitable algorithm to give a new modified metadata Mi. 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.

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

For the metadata (mi) of each data block the number αi is added to get a new k bit number Mi.

$M_i = m_i + \alpha_i$ (3)

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

D. Appending the Metadata

All the metadata bit blocks that are generated using the above procedure are to be concatenated together. This concatenated metadata should be appended to the file F before storing it at the cloud server. The file F along with the appended metadata F is archived with the cloud.

E. Verification Phase

Let the verifier V 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 the verifier accepts or rejects the integrity proof. Suppose the verifier wishes to check the integrity of nth block. The verifier 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 verifier knows. The verifier also specifies the position at which the metadata corresponding the block i is appended. This metadata will be a k-bit number. Hence the cloud storage server is required to send k+1 bits for verification by the client. The metadata sent by the cloud is decrypted by using the number αi and the corresponding bit in this decrypted metadata 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 client's data at the cloud storage.

IV. Conclusion

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. 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. But in our scheme the archive just need to fetch and send few bits of data to the client.

V. 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

[4]Muralikrishnan Ramane and Bharath Elangovan., "A Metadata Verification Scheme for Data Auditing in Cloud Environment: International Journal on Cloud Computing: Services and Architecture (IJCCSA), Vol.2, No.4, August 2012

[5]Aderemi A. Atayeroq and Oluwaseyi Feyisetan(2011),"Security Issues in Cloud Computing: The Potentials of Homomorphic Encryption", Journal of Emerging Trends in Computing and Information Sciences ,Vol. 2, No. 10,pp.546.

