An Approach to Minimize Computational and Communicational Overhead in Cloud Computing

Kalpana Batra
Student M.Tech , Dept. of CSE,
SGT Institute of Engineering and Technology,
Gurgaon, India

Abstract- Cloud computing has been envisioned as the next generation information technology architecture for enterprises. Cloud Computing moves the data on the cloud storage servers maintained by service providers, which deprive the user of their control of the physical possession of data, even though they are the owners of the data. This unique feature however has raised many new security challenges which have not been well understood. Efficient verification techniques are of significant importance for cloud customers to validate data integrity and availability are not applicable on the outsourced data without having a local copy of the data. Downloading all outsourced data in order to validate its integrity is impractical for the excessive I/O cost and the high communication overhead across the network Therefore efficient techniques are needed for this. In this paper, we introduce a framework and efficient constructions for dynamic provable data possession (DPDP), which extends the PDP model to support provable updates on the stored data.

Keywords: Cloud Computing; Integrity, PDP, DPDP.

I. INTRODUCTION

Cloud computing is an internet based computing. It dynamically delivers everything as a service over the Internet based on user demand, such as network, operating system, storage, hardware, software, and resources. These services are classified into three types: Infrastructure as a Service (laaS), Platform as a Service (PaaS) and Software as a Service (SaaS). Cloud computing is deployed as three models such as Public, Private, and Hybrid clouds [1]. Moving data into the cloud offers great convenience to users since they don't have to care about the complexities of direct hardware management. In this new model, the users put their data on the cloud storage servers maintained by service providers, which deprives the users of their control of the physical possession of data, even though they are the owners of the data. In this case, some new security [10] needs and problems have arisen. At the same time, when one's data are outsourced, he wants to know whether the data is truly stored at the correct servers and be intact as stated in the Service Level Agreement (SIA). Efficient verification techniques are of significant importance for cloud customers to validate data integrity and availability are not applicable on the outsourced data without having a local copy of the data Downloading all outsourced data in order to validate its integrity is impractical for the excessive I/O cost and the high communication overhead across the network Therefore efficient techniques are needed for this.

The Provable Data Possession (PDP)[2] scheme is introduced to ensure the integrity of data in a typical PDP model, the data owner generates some metadata/information for a data file to be used later for verification purposes through a challenge response protocol with the remote/cloud server. The owner sends the file to be stored on a remote server which may be untrusted, and deletes the local copy of the file.

In this paper, we introduce a framework and efficient constructions for dynamic provable data possession (DPDP), which extends the PDP model to support provable updates on the stored data. Given a file F consisting of n blocks, we define an update as either insertion of a new block (anywhere in the file, not only append), or modification of an existing block, or deletion of any block Therefore our update operation describes the most general form of modifications which a client may wish to perform on a file.

From literature survey, it has been observed that there is PDP schemes for multiple copies of a file, but it is for static data. We propose dynamic multiple copy provable data possession scheme. Through this scheme, it supports outsourcing of data with adequate guarantee. This scheme supports the dynamic operations[9] like modification, insertion, deletion and append. The data owners & authorized users have access to the files store on the cloud server. This scheme provides the security (correctness) against colluding servers. Cloud servers can provide valid responses to verifier's challenges only if they actually have all data copies in an uncorrupted and updated state. This helps to reduce the communication and computational complexity.

The rest of the paper is organized as follows. Section 2 describes the related work. In section 3 we have proposed scheme. In Section 4 we have conclude our paper.

II RELATED WORK

Provable Data Possession (PDP) [2] is the scheme for validating integrity for the outsourcing data storage. Definition of PDP model is first proposed by Ateniese et al. He also describes the related operations and functions. So that owner can verify the data file at any time or periodically. A simple solution is to fragment the file into data blocks and generate the Message authentication code for each block. Both the file and MAC is sending to cloud server by deleting the original file. But keep the private key with them to verify the data blocks. Along with MAC and private key, data blocks can be retrieved to verify the randomly selected data blocks. But this suffers from various drawbacks, as communication latency is linear to the data size that is queried [7]. Juels et al. [6] proposed a formal definition of POR and its security model. In their work, they divide the file into small data blocks after encryption. This file is encoded by Reed-Soloman codes. The data file is added into some "sentinels" to detect whether it was intact. But both schemes[2][6] do not support dynamic data update and can only verify limited times because that the two schemes only have finite number of the "sentinels" in a file. Number of PDP protocols [2][3][4][5] have been proposed but all these protocols supports to static data. The application which requires frequent change in data or in file cannot support by these protocols.

III DYNAMIC PDP SCHEME

A. Cloud Data Storage Model

We consider a cloud data storage service involving three different entities, as illustrated in Figure l: the Data Owner, who has large amount of data files to be stored in the cloud. r; the cloud server (CS), which is managed by cloud service provider (CSP) to provide data storage service and has significant storage space and computation resources (we will not differentiate CS and CSP hereafter.); the authorized users, who are the clients of data owners. Authorized users have access to use the data owner's files. Authorized users make the request to CS, CS returns data in the encrypted form. These encrypted files are decrypted by using the private key of the authorized users. These private key is already shared between data owner and authorized user.

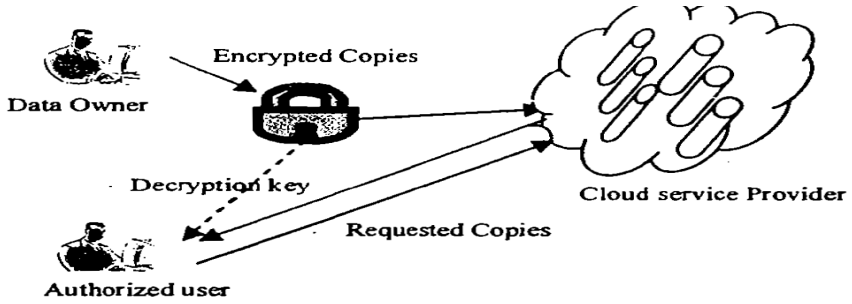

Figure1. Cloud Data Storage Architecture

Here we assume that the interaction between the owner and the authorized users to authenticate their identities and share the secret key has already been completed, and it is not considered in this paper.

B. Notations

F - the data file[8] to be stored, we assume that F can be denoted as a matrix of m equal-sized data vectors, each consisting of l blocks, these all data blocks belongs to Galois Field GF(2^P) where p= 8 or 16 or 32. Let F = (F_1, F_2,F_3,…..F_m) and F_l = (F_{1i} , F_{2i} ,F_{3i} ,…..F_{li})^T where (i ε s{1….m}), where i < = 2^P-1. File distribution is a symmetric layout with parity vectors are achieved with the data distributed matrix 'X' derived from Vander monde Matrix.

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ b_1 & b_2 & \cdots & b_n \\ b_1^2 & b_2^2 & \cdots & b_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ b_1^{n-1} & b_2^{n-1} & \cdots & b_n^{n-1} \end{bmatrix}$$

Where b_j (j ε (1…k)) are distinct elements randomly picked from GF (2^P).

1. f_{key}(.) – Pseudorandom Function defined as F: ({0, 1} * x key – GF(2^P)
2. Φkey(.) – Pseudorandom permutation which is defined as ∏: [0, 1] log2$^{(l)}$ x key-{0 , 1}log$^{2(1)}$
3. #(.) cryptographic hash function

C. MAP TABLE DPDP (MT - DPDP)

Map table is stored on site of owner's or authorized users. It is a small data structure stored, which

has 3 fields: Sequence Number (SN), Block Number (BN) and Version Number (VN). Map table is shown in figure 2. File is divided into blocks. SN is an indexing on the file block. It indicates the physical location of the file block. The BN is a counter used to make a logical numbering/indexing to the file blocks. Thus, the relation between BN and SN can be viewed as a mapping between the logical number BN and the physical position SN. The VN indicates the current version of file blocks. When a data file is initially created the VN of all blocks is 1. If a specific block is being updated, its VN is incremented by 1. It is of significant importance to note that the verifier keeps only one table for any arbitrary number of file copies.

D. *Procedural Steps*

- *Key Generation* ($key_{pub,}$ $key_{priv,)}$: This is the algorithm run by the authorized user. ($key_{pub,}$ $key_{priv,)}$ is a probabilistic algorithm run by the user. It takes as input a security parameter, and outputs a private key key_{pri} and a public key key_{pub}. The client stores the private and public keys, and sends the public key to the server.
- *File Distribute ($key_{pub,}$ $key_{priv,}$, F):* This is an algorithm run by the data owner to prepare (a part of) the file for untrusted storage. It takes the input as private key_{pri} and public keys key_{pub} (a part of) the file F. It generates number of copies F_i of a file F. A file copy F_i is an ordered collection of blocks $\{b_ij\}$ $1 \leq j \leq m$.
- *Tag Gen* generates the tag set Φ. The tags are generated for each block of file. The set Φ is an ordered collection of tags for the data blocks σ_{ij} is the tag generated for the block bij. σ_{ij} is generated using the hash function on BN_i and VN_j and the unique file identifier Fid which identifies the users file. This algorithm is run by the data owner. It takes parameters as tag set Φ, file id F_{id} and the file F.
- *Prepare _update* is the algorithm which is run by data owner to do dynamic operation like delete, insert the contents, or modify the block. Prepare_update generates the updated information.
- *Modifications:* Suppose the file F = $\{b_1, b_2, b_n\}$. If the owner or authorized user wants to modify the block b_j with bj' for all copies of F. This Prepare update algorithm has to do the following thing:

  1. VN_j should be incremented by 1.

  2. Update the block b_j with $b_{j'}$. With this modification block size may increase or decrease. As per the block size, adjust the block with creating a new or deleting the block.

  3. Generate new tag σ'ij for this block b_{ij} using hash function on BN_j and VN_j and the unique file identifier

  4. User send modify request to CSP with parameters F_{id}, $b_{j'}$, σ'ij.

- *Exec Update* is the algorithm run by CSP. CSP takes the input as file copy $F_{i,}$ σ'_{ij}. After receiving this request CSP does the following things:

  1. Update the block b_j with $b_{j'}$ along all copies $F_{i.}$

  2. Replace the tags σ'_{ij} for σ_{ij} in the set Φ.

- *Insertion:* Ins_Block is the algorithm run by the user to insert a new block b' after the position j in all file copies.. Prepare_update algorithm has to do the following thing:

  1. Enter the new entry into Map table with fields SN, BN, and VN. If Map table has n number of SN values then SN_{n+1} = n+1, VN has the initial value 1. BN has the value max(BNj) $_{1<j<m}$+1

  2. For block b', create new tag σ'i by applying the hash function on BN_j and VN_i and the unique file identifier.

  3. Ins_Block sends the insert request to the CSP of insert block with the information of b', σ'j, Fid.

Upon receiving an insert request, CSP runs the ExecUpdate algorithm. CSP does the following things:

1. Inserts the new block b' in all file copies F_i for the unique Fi_d.
2. Inserts the tag σ'_j at the j^{th} position in the set Φ.

- *Deletion :* Del_Block algorithm is run at user site to delete the block at position j in the Map Table. Del_Block sends the request to CSP with the information of b_i , file id F_{id}. It deletes the block from all file copies F_i. Upon receiving this request the CSP does the following things:

  1. It deletes the block at position j from all file copies $F_{i(1<i<n)}$

  2. Delete the tag σ_j from tag set Φ for the deleted block $b_{j.}$

The Figure 2 shows various changes due to dynamic operations. Figure 2.a shows initial Map Table consists of SN, BN, and VN. There are *five* blocks and VN has initial value 1. Figure 2.b shows modifying the block at position 3. The value of VN has changed 2. Figure 2.c shows deleting block at position 2. In these remaining subsequent blocks is shifts one position upwards. The SN has not changed during dynamic operations as it shows the physical location of the block.

SN	BN	VN
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1

Fig 2.a

SN	BN	VN
1	1	1
2	2	1
3	3	2
4	4	1
5	5	1

Fig 2.b

SN	BN	VN
1	1	1
2	3	1
3	4	2
4	5	1

Fig 2.c

SN	BN	VN
1	1	1
2	2	1
3	6	1
4	3	2
5	4	1
6	5	1

Fig 2.d

Initial Map table	Modifying Block at position 3	Deleting Block at position 2	Insert Block at Position 3

Figure2. Change in Map Table due to Dynamic Operation

Challenge Response Protocol: Challenge Response Protocol is used to verify the correctness and integrity of all file copies of file.

Prove is the algorithm run by CSP and outputs proof P as a guarantee to actually storing of n number of file copies which are updated and consistent. This algorithm takes input as file copy F_i and tag set Φ.

Verify is the algorithm run by the data owner to check the correctness of the file copies. It outputs 1 if all files are verified correctly otherwise 0. It takes the input as proof P return by the CSP and public key key_{pub}.

Challenge Response Protocol has the following steps:

1. Verifier sends c number of blocks to be challenged and two challenge keys pseudorandom function and pseudorandom permutation, $f_{key}(.)$ with k_1 and Φkey (.) with k_2 respectively. These two keys are used to generate a set Q = [(j r_j)] pair of random indices and random values.
2. CSP generates a set Q as the verifier did. It computes the σ. After generating the set Q = {(j,r_i)) of random indices and values, the CSP runs the Prove algorithm to generate evidence that it still correctly possesses the n copies in an updated and consistent state. CSP responds with proof P having a set of {σ , μ}.
3. Upon receiving the Proof, verifier runs the Verify algorithm to check the verification equation.

The verify algorithm returns 1 when data is not changed, otherwise 0.

IV ERROR LOCALIZATION

The verification equation will be true only if all the copies of files are correct and consistent if the proof P = {σ, μ} generated by CSP will be valid. When one or more copies are corrupted then whole auditing procedure fails. To respond this situation, data owner generates the tag σ_{ij} for the block b_{ij}. But aggregations of tags are not done for the blocks at the same indices in each copy. Therefore the tag set Φ becomes $\Phi = \{\sigma_{ij}\}1 \le j \le n, 1 \le j \le m$.

Upon receiving the proof P, the verifier equation validates this P. If the verification equation returns true means all file copies are intact, correct and consistent. But if verification equation fails, the verifier ask CSP to send $\sigma = \{\sigma_i\}_{1 \le i \le n}$

V PERFORMANCE ANALYSIS

In Map table, each entry requires 8 bytes of storage. Thus the total number of entries equals to the total number of file blocks. During implementation, there is no need to store SN in Map table. However, there is only one table for all file copies which increases the overhead for storage on verifier side. Therefore the storage complexity are O(m) and O(mn) for Map table and the n copies. For example, if we are dealing with 64MB file with block size 4KB, the size of the map table will be only 128KB for any arbitrary number of copies (table size = 2MB for 1GB file).

It reduces 83% of the computation time of cloud server provider. And Verification time can be reduced up to 95%. In Map table, the storage overhead is independent of the number of copies.

VI CONCLUSION

As the development of cloud computing, security issue has become a top priority. This work represents a new paradigm of security in cloud. Security is the responsibility of all parties involved in laaS cloud computing. Vendors are responsible to provide a secure fabric. Information owners are responsible to protect their data. This method achieves availability, reliability & integrity of data in cloud storage. This scheme supports to the dynamic operations like insert, delete, append. This is better than the single copy of DPDP. Through detailed performance analysis, this scheme provides more security to user's data in cloud computing against Byzantine failure, unauthorized data modification attacks and even server colluding attacks. We believe that data storage security in Cloud Computing, an area full of challenges and of paramount importance, is still in its infancy now, and many research problems are yet to be identified.

REFERENCES

[1] I-Hsun Chuang, Syuan-Hao Li, Kuan-Chieh Huang, Yau-Hwang Kuo "An Effective Privacy Protection Scheme for Cloud Computing", ICACT-2011, pp. 260-265

[2] Ateniese, G., et al., "Provable data possession at untrusted stores", in Proceedings of the 14th ACM conference on Computer and communications security. 2007, ACM: Alexandria, Virginia, USA. p. 598-609.

[3] Y. Deswarte, J.-J. Quisquater, and A. Sa"idane, "Remote integrity checking," in 6th Working Conference on Integrity and Internal Control in Information Systems (IICIS), S. J. L Strous, Ed., 2003, pp. 1-11.

[4] D. L. G. Filho and P. S. L M. Barreto, "Demonstrating data possession and uncheatable data transfer," Cryptology ePrint Archive, Report 2006/150, 2006.

[5] P. Golfe, S. Jarecki, and I. Mironov, "Cryptographic primitives enforcing communication and storage complexity," in FCO2: Proceedings of the 6th International Conference on Financial Cryptography, Berlin, Heidelberg, 2003, pp. 120 —135.

[6] Juels, A. and J. Burton S. Kaliski, "Pors: proofs of retrievability for large files", in Proceedings of the 14th ACM. conference on Computer and communications security. 2007, ACM: Alexandria, Virginia, USA. p. 584-597.

[7] K. Zeng, "Publicly verifiable remote data integrity," in Proceedings of the 10th International Conference on Information and Communications Security, ser. ICICS '08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 419— 434.

[8] P. Syam Kumar, R. Subramanian and D. Thamizh Selvam, "Ensuring Data Storage Security in Cloud Computing using Sobol Sequence", International Conference on Parallel, Distributed and Grid Computing (PDGC 2010),pp. 217-222.

[9] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, Jin Li, "Enabling Public Auditability and Data Dynamics for Storage Security in Cloud Computing", IEEE transactions on parallel and distributed systems, vol. 22, no. 5, may 2011, pp. 847-860

[10] Minqi Zhou, Rong Zhang, Wei Xie, Weining Qian, Aoying Thou "Security and Privacy **in** Cloud Computing: A Survey", IEEE 2010 Sixth International Conference on Semantics, Knowledge and Grids, pp. 105-112